AUTOMATED LANGUAGE ANALYSIS

1967 - 1968

Report on research for the period
March 1, 1967 - February 29, 1968

The research reported herein was conducted under Contract N00014-67-A-0321, Office of Naval Research, U. S. Navy, Task No. NR 348-005

at the
Department of Information Science
University of North Carolina at
Chapel Hill

The research reported herein was conducted under Contract N00014-67-A-0321, Office of Naval Research, U. S. Navy, Task No. NR 348-005

at the
Department of Information Science
University of North Carolina at
Chapel Hill

A U T O M A T E D L A N G U A G E A N A L Y S I S

1967 - 1968

Report on research for the period
March 1, 1967 - February 29, 1968

Sally Y. Sedelow, Principal Investigator
Walter A. Sedelow, Jr., Consultant
Walter L. Smith, Consultant
Joan N. Bardez
H. William Buttelmann
William G. Hickok
Joan Peters
Terry Ruggles

Automated Language Analysis

1967 - 1968

Report on Research for the Period
March 1, 1967 - February 29, 1968

by

Sally Yeates Sedelow

ABSTRACT

This report describes current research associated with the automated language analysis project. The focus of the research is upon the delineation of patterns formed in the linguistic coding of information; this delineation is called stylistic analysis. The report describes research on thesauri, especially research upon comparisons of possible input thesauri upon methods of enlarging input thesauri, and upon the possible use of input thesauri for the resolution of semantic ambiguity. The report also describes a new ring-structure version of the VIA program which produces text-specific thesauri. Work on statistical approaches to the analysis of strings is also described. The report contains extensive documentation for the PL/1 implementations of the INDEX, SUFUN, and SUFFIX sections of the VIA program, as well as a description of the program design for the ring-structure version of VIA.

PREFACE

We would like to thank the people and organizations who have supported this research during the past year. The members of the Information Systems Branch, Office of Naval Research, have provided advice and administrative support. Various offices within the University of North Carolina at Chapel Hill, especially Research Administration and Contract Administration, have helped to manage the administration of the contract. The computation facilities at TUCC (Triangle Universities Computation Center) and at UNC, Chapel Hill, have, of course, been vital to this project. We would also like to thank for their support the academic departments at UNC, Chapel Hill, with which project personnel are associated. Special thanks go to the Information Science Department for providing the assistance necessary for publishing this report and other documents associated with this contract; Mrs. Sallie Clotfelter has been most consistently associated with this phase of the project and we wish to express our gratitude to her for her good assistance.

TABLE OF CONTENTS

Survey of the Computer-Aided Language Analysis Project,

March 1, 1967 - February 29, 1968

I. INTRODUCTION

The general goal of this research project on automated language analysis is to develop a set of computer programs which will characterize as completely as possible, the style of any string of language under examination. For this research, style is defined as "the patterns formed in the linguistic encoding of information"; style may be said to consist of patterns of word choice as well as of patterns of types of words, punctuation, groupings of words, etc. The ability to characterize style implies that one author can be distinguished from another, one sub-culture from another, one culture from another, and the like.

The utility of the capacity to make such distinctions is wide-ranging. If one author or speaker can be distinguished from another, then interpolations or modifications of a given statement which might represent shifts in policy or attitude can be detected. The distinction of cultures and sub-cultures is useful both for the information gain inherent in the process of distinction, and for the uses to which an awareness of stylistic distinctions can be put. It may well be that basic cultural, political, and military misunderstandings derive, in part, from ignorance of stylistic distinctions among communication patterns. An example of such a distinction -- in this case a recognized distinction -- is given by Jagjit Singh in his recent publication, Great Ideas in Information Theory, Language and Cybernetics. Singh points out that in Western languages, there is a "great deal of

dichotomizing; terms go in pairs, with meaning reference to opposite extremes (good-evil, black-white, clean-dirty, strong-weak and so on). In the Chinese, there is much less dichotomizing and the language has few pairs of opposites. Consequently, what may seem to be a stark contradiction to a Western mind may not be so to a Chinese."[1] The identification of such major stylistic differences, as well as more subtle ones, is, thus, of real importance for officials engaged in diplomatic efforts as well as for those concerned with the analysis of information.

Thus far our efforts to delineate style have concentrated upon patterns of word choice and the distribution of such patterns, as well as interrelationships among patterns. The two programs which have been used for this research are VIA and MAPTEXT. Thorough descriptions of the programs and of the purposes for which they have been used are available in a number of research reports and articles.[2] Briefly, VIA produces what we have called text-specific thesauri and MAPTEXT produces abstract representations of any specified linguistic element or elements. We have used MAPTEXT to accept as input, output from the VIA program in order to display graphically the spatial dynamics of word patterns in a given text. As we reported in the Third Annual Report, the use of MAPTEXT in this way facilitated the quick recognition of a major rhetorical

---

[1] Page 4, Dover Publications, Inc. New York, 1966.

[2] Stylistic Analysis, First Annual Report, SDC Document TM 1908/100/00, March 1, 1965; Stylistic Analysis, Second Annual Report, SDC Document TM 1908/200/00, March 1, 1966; Stylistic Analysis, Third Annual Report, SDC Document TM 1908/300/00, March 1, 1967; Updating of THESAUR Program, SDC Document TM 1908/009/00, December 17, 1965; "A Preface to Computational Stylistics," Computer and Literary Style, edited by Jacob Leed, Kent State University Press, 1966, pp. 1-13. Co-author with Walter A. Sedelow, Jr.; "Stylistic Analysis," Automated Language Processing: The State of the Art, edited by Harold Borko, John Wiley and Sons, 1967. Co-author with Walter A. Sedelow, Jr.

pattern having to do with references to conventional and nuclear warfare in Soviet Military Strategy.[3]

II. Work during Past Year

The project work during this past year has been concentrated upon four major areas: 1. Research on thesauri; 2. Conceptualization and design of a new procedure, based on ring structures, for clustering words in the VIA programs; 3. Re-programming VIA for the 360 system and PL/1; 4. Initiating research on statistical descriptions of string input.

A. Research on thesauri. (This section is based upon the research of Sally Y. Sedelow assisted by Joan Peters.)

The necessity for research on thesauri has evolved from our desire to automate fully the VIA program. As VIA now operates, an input text (either a written text or transcribed oral discourse) is read into the computer, indexed, sorted alphabetically, and then sorted by root. At this juncture in the program, a computer printout is provided which shows each word type appearing in the text, which shows these types grouped according to root, and which indicates where in the text each token of the word type appears. A frequency count for each root group is also provided. Given this printout, the researcher or information analyst then decides which among the root groups in the printout he would like to explore further by looking for words in the text which are semantically associated with the given root groups. Having made this decision,

---

[3] For this research we used two translations of V. D. Sokolovsky's Soviet Military Strategy: I. Dinerstein D., Goure and Wolff, Prentice-Hall, Inc. 1963; II. Translation Services Branch, Foreign Technology Division, Wright-Patterson AFB, Frederick A. Praeger, Inc. 1963.

the information analyst then manually consults textual context, thesauri, and synonym dictionaries for words which are semantically or ideationally associated with words in the root groups he has already designated. The words manually selected by the information analyst are then submitted to the next phase of the VIA program, THESAUR, which looks for these words in the text and, when they occur, links them to the appropriate root group. If any of these newly linked words happens to be a part of one of those root groups earlier designated by the information analyst as worthy of further research, the computer goes on to search the text for words associated with that root group, and so on down through five levels. These lists are based on the manually provided first level list, but through the additional levels provided by the computer words are brought together into new clusters not specified by the human researcher or information analyst. Although the manual search does not require an enormous amount of time and it may well be useful to have the option of providing words manually, it is nonetheless desirable to eliminate the necessity of a manual search. The only way it can be eliminated is to have available a thesaurus which the computer can consult after the text has received its initial sorts and frequency counts.

Recognizing the need for a thesaurus presents no difficulty, but providing the thesaurus is another matter. An enormous amount of work has already gone into special purpose thesauri used for information retrieval of medical information, legal information, chemical information, etc. Our goals, however, require an all-encompassing, general-purpose thesaurus. The existing thesauri which come closest to being general purpose are, of course, the various versions of Roget's Thesaurus. Accordingly, we have conducted research on two versions of Roget's Thesaurus, but we have also initiated research on ways of altering and enlarging existing thesauri. One other related research

project involving thesauri has also been undertaken. This project is an exploration of the possibility of using a thesaurus, once available, in computer-accessible form to resolve or help resolve semantic ambiguities.

An initial effort directed toward the comparison of thesauri was described in last year's annual report. Comparisons of the results of VIA runs using, respectively, Webster's Dictionary of Synonyms, Roget's University Thesaurus, and Roget's International Thesaurus, suggested that the output from the runs using words from the Dictionary of Synonyms would be too spare to be useful, whereas the output from runs using Roget's University Thesaurus was too inclusive to be useful. For example, the University Thesaurus linked such words as analysis, page, metric, and line to a category headed by the word dead. Among these three reference works, the International Thesaurus seemed to produce the most useful output. The output from runs using the International Thesaurus was considerably more comprehensive than that produced by Webster's Dictionary of Synonyms, but it did not include the large number of seemingly extraneous words produced by the runs using the University Thesaurus. The results described in the report on the third year of research were, however, highly tentative because just a few words (especially DEAD and DECLINE) had been examined in exhaustive detail. To indicate the magnitude of the task, it is worth noting that Roget's University Thesaurus produced a list of 2,452 words, (using the word DEAD as the primary or key word) for VIA's operation on a chapter of Soviet Military Strategy; Roget's International Thesaurus produced 268 words under the category, DEAD. This past summer, we examined VIA's output for comparative runs on twelve other words: DECREE, DEFLECT, DELIBERATE, DELUSION, DEVELOP, DEVISE, DISARRAY, DISASTER, DISTRICT, DOUBLE, and DRILL. Our investigations provide our earlier conclusions with firmer support.

Differences both in formating and indexing between the University and International Thesauri account most heavily for the discrepancies between the outputs of VIA runs using, respectively, the University and International Thesauri. The University Thesaurus does not subsection categories as does the International Thesaurus. For example, the index for the word DEATH in the University Thesaurus refers to a category which includes both nouns and verbs in a great range of meanings. The index for DEATH in the International Thesaurus sub-divides on the basis of meaning and syntax; the indexed section for DEATH as a noun in the International Thesaurus contains 26 entries as opposed to the single all-inclusive category of about 221 entries in the University Thesaurus. Thus the format, itself, of the University Thesaurus, results in a wider retrieval net. The indices of the two thesauri reflect the difference in formating. In the International Thesaurus, a typical entry might read "lifeless 407.24". The analogous entry in the University Thesaurus is "lifeless 360." Clearly, the International Thesaurus index is referring to a sub-section (.24) of a main entry (407; the entry in this case happens to be DEAD) while the index in the University Thesaurus is referring to a complete category.

In addition to these format-defined differences between the indexing methods of the two thesauri, there are many reflections of the eclecticism of the editors of those particular editions. Our work this summer again supported our earlier observation that the International Thesaurus tends toward indexing on more abstract terms, while the University Thesaurus includes terms which are rather more precisely focused, often upon human manipulation. For example, under the word DECREE, the International Thesaurus refers to the headings JUDGMENT, COMMAND, and LEGALITY; the University Thesaurus refers to these words as headings, but also to the heading, LAWSUIT. Or, as another

example, the International Thesaurus under DELUSIONS, has ALLUSION, DECEPTION, PSYCHOLOGY, AND PSYCHOTHERAPY; the University Thesaurus refers to INSANITY and SELF-CREDULOUS. Furthermore, it is rather often the case that the index entries which do occur in common in both thesauri, do not refer to the same words in the thesaurus proper. For example, although the heading is indexed in both thesauri under DEVELOP, the word ADVANCE is indexed by EVOLUTION in the International Thesaurus but not in the University Thesaurus. Rather, the word ADVANCE is indexed by the heading INCREASE in the University Thesaurus. In summary, we might note that our examination of VIA runs turned up 34 words which occur in both thesauri and could have appeared under a common heading in both thesauri, but instead occurred under disparate headings. Perhaps it is also worth underlining the obvious here -- the number of words for which we had comparative VIA runs totaled just 14. The disparities between the headings indicated by the indices would doubtless seem overwhelming were a more extensive study undertaken.

There seem to be several other important distinctions between the two thesauri. For example, any given word in the International Thesaurus is likely to appear in more categories than it does in the University Thesaurus. Furthermore, the categories in which the word might appear in the University Thesaurus may seem of more limited scope than the categories in the International Thesaurus (e.g. a word which might appear under the heading LAWSUIT in the University Thesaurus as opposed to LEGALITY in the International Thesaurus.) Thus, the International Thesaurus seems to emphasize more diversified, and at the same time, more general denotations of words than does the University Thesaurus. The index of the University Thesaurus does not even contain certain rather common word forms, such as CONCEPT.

Our experiments with these thesauri and the Dictionary of Synonyms were constrained by the need to use computer time efficiently and therefore, by some concomitant research requirements. Hence, the words chosen for the comparative runs were words appearing in one translation, and not the other, of Soviet Military Strategy, because we were looking for patterns of word choice which might distinguish one translation from the other. These constraints notwithstanding, it would seem apparent that if there were a serious interest in using the International Thesaurus as a basis for a general-purpose, computer-accessible thesaurus, a rather thorough study of indexing technique and indexing bias would be appropriate. We intend to make some further effort in that direction manually, but a comprehensive study demands that the International Thesaurus be available in computer-accessible form. The cross referencing systems used in a thesaurus could then be explored in considerable depth.

In the absence of a computer-accessible general thesaurus, indexing bias could be further explored with the help of the VIA program. For example, one might designate a certain number of primary words used in disparate types of text (e.g. the word STATE as used in Soviet Military Strategy, Hamlet, etc.) to see just what the retrieval net of a given indexing term seems to be. If one indexing category tends to pick up a large number of other categories, or primary words, the degree of relevance of the primary words thus retrieved to the main indexing category might be investigated with particular care. Such an approach would have the advantage of moving toward an operational definition of indexing bias through a comparative testing and examination of groups of words retrieved from any given text.

Our exploration of possible ways of interplaying a textual word, its context, and a thesaurus, to enrich a thesaurus have really just begun. The

task of building a thesaurus is so onerous and time-demanding that it would seem desirable to find a way of using an interactive computer program to assist with the enlarging of a thesaurus. Enlarging such a thesaurus might take two directions: 1. the addition of words to a thesaurus base which is used for the examination of any text; 2. the provisional addition of technical terms to categories or, indeed, the addition of technical categories as well, to a thesaurus, with the provision that such words would be used just for retrieval purposes in appropriate technical areas. The desirability of having a general purpose thesaurus to which one can add one or several technical thesauri has been demonstrated by our own work on Soviet Military Strategy. There are many rather technical terms in Soviet Military Strategy having to do with methods of warfare. It would be useful to have a technical thesaurus to cope with the many special terms connected with, for example, nuclear warfare. A thesaurus such as Roget's International Thesaurus is rather strong on a value, attitudinal vocabulary, but relatively weak on a technical, scientific vocabulary. If one could have consistent access to the former vocabulary and access, at need, to the latter vocabulary, one would have a very strong research tool for the analysis of content, idea, etc.

We are interested in seeing whether when a given word does not appear in a thesaurus a word in its immediate context does appear and, if so, whether the originally specified word can be added to the thesaurus category of the context word. An example, might be the word ICBM in the context, ICBM MISSILE. If the word MISSILE already appeared in the thesaurus, ICBM might be added to that category for special-purpose retrieval. Joan Peters has begun an exploration of such possibilities. She has examined the permuted and subject index to Computing Review for 1964-65 for terms from the areas of automatic control, electronics, and mathematics, which might, or might not

appear in Roget's International Thesaurus. She has then investigated the thesaurus to see which among these words do, indeed, already appear in the thesaurus; next, she has looked up the dictionary definitions for each of the words. This work has resulted in two lists: first, a list of words that occurred in the International Thesaurus but needed to be added to other categories because, for example, their automatic control denotations were not included in the International Thesaurus and second, a list of words that occurred nowhere in the International Thesaurus. Next she will go to the context of the words on her list to see whether words in the context would help classify and place the unlisted words into appropriate categories in the International Thesaurus. If she is able to find good leads in the context, the next problem is to locate cues which would enable this entire process to be automated. A workable solution, if this approach seems at all promising, may be to use a graphic display which would enable a human being to make the final decisions as to the classification of a given word.

Another related research project is our effort to determine whether a computer-accessible thesaurus might help in the resolution of ambiguities. The question here is whether for words such as STATE, which have multiple meanings, it would be possible to interplay textual context and thesaurus entries to resolve ambiguities without resorting to a parser. Or, if a parser were required, could it be a simplified version? Or, if an elaborate parser were required, could the text-thesaurus interplay make ambiguity resolution much better than any automated procedure currently in existence?

So far as the VIA output is concerned, the problem of ambiguity arises

when a word such as STATE appears in a cluster of words. Sometimes it is possible to determine rather accurately the meaning of an ambiguous word by simply looking at the words associated with it in the output. For example, if the word GENERAL appeared and the words associated with it were COMMON, USUALLY, and WIDESPREAD, one might assume that GENERAL did not have military connotations in the given text, but rather was used in some such phrase as IN GENERAL. The output, however, does not always resolve ambiguities and we have been considering ways of solving this problem. The fastest and most efficient way to deal with words of very low frequency would simply be to print out the context for such words. Supplying contexts for words of very high frequency, however, is not so efficient. It takes a good deal of time, relatively speaking, to examine the context for two hundred occurrences of a given word type. One method of using a thesaurus to deal with high frequency ambiguous words might be to have the computer program examine the context of an ambiguous word and use the thesaurus to establish the part of speech for the contextual words. For example, if the word STATE followed MILITARY and the phrase MILITARY STATE were followed by a verb such as DEMANDED, STATE would pretty clearly be used as a noun. For words of very high frequency, the program might print out a summary of contextual frames. For example, the frame, adjective STATE verb, might occur 100 times. Such frames, combined with the associated words appearing in the output, might provide all the information needed in some instances -- most notably, those instances in which the major ambiguity coincides with a part of speech distinction, e.g., a word such as SHIP, which has one main denotation as a noun and a quite different denotation as a verb. In many cases, however, as noted in earlier research reports for the project, semantic ambiguity does not necessarily coincide with syntactic ambiguity. For example, the word AID has much the same meaning

when used either as a noun or a verb. For purposes of content or information analysis, such syntactic ambiguity is not of major importance. Words associated with AID, used either as a noun or verb, would very probably be highly similar.

Syntactic frames, therefore, although they would have the advantage over a straight parser of being based upon a specific text and therefore, very probably, not exhibiting all the possible syntactic frames a parser would produce[4] would not seem to solve such semantic ambiguities as STATE used in the phrases MILITARY STATE or SOVIET SOCIALIST STATE as opposed to its use in the phrases, STATE OF MIND or STATE OF A NATION. To solve this latter kind of problem it may well be that a combination of grammar rules and a thesaurus or a printout of context types would be most helpful. An example of a context type would simply be the phrase STATE OF THE. If the word STATE were being used to mean condition, it might very often occur in that specific context. Therefore, the context would printed out just once with an accompanying frequency of occurrences of that type. Or, one could again use syntactic frames, grouping all occurrences of the word STATE followed by a preposition followed by an article (e.g. STATE OF THE, STATE OF A,) or of a preposition followed by a pronoun (STATE OF HER, STATE OF HIS). For these syntactic frames, the thesaurus, alone, would not be adequate because thesauri do not include so-called function words. Nor would one wish to burden a thesaurus with such words. Rather, one would use a separate dictionary of the approximately three hundred and fifty such words which serve as syntactic markers in a sentence.

---

[4] Cf. the comments on syntactic rigidity on pages 62 - 68 of the Second Annual Report.

We have collected a good deal of empirical data as to what sorts of frames will resolve the ambiguity of specific words. Before operational generalizations can be made about this data, more research should be directed toward determining syntactic and semantic rigidities in different kinds of texts. By rigidities, we mean that once a word is used as a certain part of speech or for a given meaning, it tends to be used that way repeatedly in a given text. We have found very strong evidence of such rigidity in Soviet Military Strategy and Gerald Salton has spoken of such rigidities[5] in the technical literature which has served as a test for his SMART System. It would seem possible, for example, that a computer program could be so designed that for a highly rigid usage (for example, the word ARMED in the phrase ARMED FORCES in Soviet Military Strategy) the most informative and efficient procedure would be simply to print out the word within its context type. For more varied uses, syntactic contexts might be preferable. We plan, next, to direct our attention to the question of rigidity in less technical documents and texts, and we will then make an effort to generalize our empirical data so that we may test some of the general theories outlined above.

---

[5] In remarks made at the National Conference on Content Analysis, The University of Pennsylvania, November 16-18, 1967.

B. Ring-Structure Version of VIA. (By William Buttelmann)

This section describes a second version of VIA (Verbally Indexed Associations), a thesaurus-building program. The initial version of VIA is described in TM 1908/100/00, Stylistic Analysis: First Annual Report, 1 March 1965, and TM 1908/009/00, Updating of THESAUR Program, 17 December, 1965.

This version of VIA incorporates two major technological changes in the system structure. First, the thesaurus is organized as a ring structure, instead of the previous tree structure. For the notion of the use of ring structures for natural language analysis, we are indebted to the DEACON Project.[6] The ring structure is more general than the tree and is precisely the structure of current printed thesauri, whereas the tree structure is an approximation to it. Second, the programs are written to take advantage of the large data file random-access capabilities of third generation computers. This means 1) that they are designed to operate on a very large text, with a very large thesaurus (on the order of 1 million entries) and 2) that the text-analyses, and thesaurus searches and constructions have been designed with flexibility of searching in mind (e.g. one may use the system to look for content relationships in the text, either with other words in the text or with content categories in the thesaurus, but not in the text; and one may use the system to generate microthesauri specific to a given text - the latter capability belongs also to the earlier VIA; the former does not.). Finally, this system has been built with an on-line time-shared version in view.

---

[6] James A. Craig, Susan C. Berezner, Homer C. Carney, and Christopher R. Longyear, "DEACON: Direct English Access and Control," in Proceedings Fall Joint Computer Conference, 1966, pp. 365-380.

All the capabilities of the earlier VIA remain. In addition, this version has the ability always to print the words actually occurring in the text, even though they do not appear in the thesaurus, so that the comment (DIFFERENT FORM APPEARS IN THE TEXT) will never appear in the output.

General Description of Systems Programs

The new version of VIA is structured in three sections: "index"; "data preparation"; "thesaurus search and construction". The programs are written in PL/1, and were developed on the IBM S/360 Model 75. Since PL/1 is a problem-oriented language which is designed to be machine independent, the system can presumably be run on any machine configuration which has a PL/1 compiler and a random-access memory extension (such as a disk). Alternately, it can be run with a more limited thesaurus (on the order of one to ten thousand words) in a machine of appropriate memory capacity.

1. Index

The first main section of the program formats the input text into variable length records, each consisting of one textual word and index information showing where the word in the text occurred. The input text is acceptable in stream form.

2. Data Preparation

This section prepares the text and the thesaurus for the thesaurus search-and-print done in Section 3. In addition, analysis requests are processed here, and a table of search keys which initiate the searches in Section 3 is built.

The first phase, SORDIT, sorts the text records into alphabeti-

cal order. The remainder of the section is a program that performs a number of functions more or less in parallel. Function words in the text are deleted. Roots of content words in the text and in the thesaurus are identified and words with the same root are matched. Text words not in the thesaurus are added to the thesaurus. Analysis requests specifying parameters for thesaurus-text searches are batched and sorted on type. The requests are examined for any types which require word frequency counts, and if they are found the counting is done. The counts are entered in the thesaurus, so that subsequent passes of the text are not needed. Finally, the list of analysis requests are processed, and from them a table of search keys is generated. This table contains key words that initiate thesaurus searches and/or microthesaurus construction. For each section of text, the key word table is updated; thus, new key words are identified by section. In the thesaurus, a record is kept noting the text section when a word first appears; thus, new content relationships are identified by section. The key word table and the word appearance record in the thesaurus provide the information for building the microthesaurus.

The algorithms for this program are described in detail in Part III of this report. In particular, the root-identifying routines comprise the program, SUFFIX, described in the First Annual Report; they are used unchanged, except for reprogramming into PL/1.

Much of what was done in the previous THESAUR program is now done here.

One of the most important changes in the new version of VIA is the increased power of the system to identify statistically significant thematic content, chiefly because of the ring structure of the thesaurus. The primitive form of the first VIA identified significant content by simple word count. A quick sophistication of this method allowed the detection of words with the same root through the development of the match count (MATCNT) by the SUFFIX program. This made it possible to identify significance on the basis of the sum of the frequencies of all words with the same root. More often however, thematic sameness is a broader classification than root equivalence: words with different roots may signal the same theme; words with the same root may signal different themes. The relationships depend both on the orientation of the author of the text and the viewpoint of the analyst. One may wish to vary his choice of such relationships from analysis to analysis. The specification of thematic similarity is precisely what the thesaurus categories are intended to do. Given the rings of categories, it is now possible to identify significant thematic content based on the total of frequencies of all words in each category.

The obvious advantage of a category-based count over a root-based count is that significant content may be the accumulative effect of the occurrences of several roots, no one of which occurs frequently enough to be detected by the root-based method. The disadvantage is the cost of extra processing time on the computer. (Of course, it is not necessary that the categories in the master thesaurus be formed on the basis of thematic criteria. The categories of the master thesaurus are <u>a priori</u> to any text analysis, and one may choose a thesaurus organized any way he likes.)

A significant improvement in the counting mechanism would be to use

frequencies relative to normal usage. This requires tables of words and their frequencies, tabulated from random samples of the language taken from a very large population. Regrettably such tables are not, in general, available.

The capabilities of the earlier VIA are retained in this version. Since the significant themes are the basis for the choice of key words that cue the thesaurus searches, there is now considerable flexibility in the methods for choosing key words. The following is a summary of the four methods available:

Type 1 - (Frequency by category). Each category is examined. If the total of the frequencies of all words in the category exceeds a specified threshold, then each word in the category becomes a key word.

Type 2 - (Frequency by root). Each root is counted. If the total of the frequencies of all the words having that root exceeds a specified threshold, then each word having that root becomes a key word.

Type 3 - (Category). The analyst may simply designate a category as a key. Each word in the category then becomes a key word. No frequency considerations are made. This kind of search is relatively fast and can be used to look for all words related to a particular theme, whether or not the theme is represented with high frequency in the text.

Type 4 - (Word). The analyst may simply designate a key word. No frequency considerations apply. This kind of search is very fast and can be used to look for all words related to a particular word, whether or not the word occurs frequently in the text.

The reader should investigate the combinations of four analysis types and five search modes to become aware of the possibilities available.

3. Thesaurus Search and Construction

This section is the program THESAUR which searches the thesaurus for content-word and word-category relationships keyed by the table of search keys. All the textual information needed to direct the search has been entered into the thesaurus by the data preparation section, so no further references to the text file are needed.

THESAUR consists chiefly of a recursive subprocedure which, keyed by a key word from the table, searches through the thesaurus to find all the semantic categories containing the key word. Thus, in each category, it uses each related word to key another level of search. This process occurs through five levels of searching. Because of the cyclic, or "ring" structure of the thesaurus, certain redundancies are inevitable. For example, every word is related to itself. More intricately, two or more words in several categories together will cause the search to return eventually to the first category and thus repeat itself. Such redundancies are recognized by the program and suppressed.

The resulting printed output is equivalent in appearance to that of the earlier version of VIA, except that great flexibility is allowed in the kind of searching (and consequent microthesaurus construction) that is done. This flexibility comes from the fact that the base thesaurus contains many words not in the text. Some of these, however, are related to words in the text, and it may be important to know them. Thus, for example, it is possible to request a search for all words related to a given key word, even though the key word itself is not in the text at all. Also, for example, it is possible to ask for words in the text that are related in the thesaurus,

even if the words that establish the link are themselves not in the text. These options are controlled by the search parameters specified in the analysis request cards. (See pp. 93-97 for the format of the ANALYSIS Requests.) To describe fully the search options available, it is first necessary to describe the printout in more technical language:

The shape of the printed output for a search based on a single key word is called a "tree" . The key word is the "root" and the words at the lowest level of search (at the farthest right on the paper) are called "leaves". Each word or semantic category that is used as a branch point for further searching is called a "node". The root and leaves are also nodes.

There are, then, five search modes one may choose:

a. Text limited -All nodes are in the text:

As the program searches down a path of related words, it will abandon that path when it encounters the first word not in the text.

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 which are in the text. The path is printed only through the last textual word. Thus, it becomes a leaf. Intermediate nodes may or may not be in the text.

c. Text rooted - Root in the text:

Only the root is required to be in the text. All other modes may or may not be. Each path is printed down through five levels. This kind of search would be used to

look for words, in or out of the text, that are related to words in the text

d. Text related - Leaves in the text:

This mode is similar to b, except that the root is not required to be in the text. This kind of search would be used to look for words in the text related to a given word, whether or not it is in the text.

e. Thesaurus - Whole subthesaurus:

Nothing is required to be in the text. Thus, the whole section of the thesaurus related to the key word is printed.

In all modes (except a) words not in the text are enclosed in parentheses.

C. VIA Reprogramming

The replacement of the Philco 2000 at the System Development Corporation, the original home of this contract, by computers in the IBM System 360, and the projected transfer, now completed, of the contract to the University of North Carolina, which also has System 360 computers, necessitated rewriting the programs used for this research. Because we had wanted the programs to be accessible to as many research scientists and scholars as possible, we had used FORTRAN, insofar as possible, in the initial versions of the programs. FORTRAN is by no means an ideal language for the manipulation of verbal data, but we assumed it would be improved to make up for its deficiencies in that area. The announcement of the development of PL/1, which does have good facilities for manipulating verbal data, combined with the prospect of little improvement in FORTRAN in this regard, resulted in our decision to shift to PL/1 as the programming language for the language

analysis programs. The use of PL/1 means that we do not have to shift to a machine-oriented language for character manipulation; using FORTRAN on the Philco 2000 we were forced to write significant subsections of the programs in TAC, the Philco 2000 assembly language. The decision to switch to PL/1 resulted in a delay in making the programs operational because of the considerable slippage in the release of PL/1 compilers; other difficulties were experienced because the early releases of PL/1 represented considerable modification of the initial descriptions of the language.

Terry Ruggles of the System Development Corporation and William Hickok of the Information Science Department at the University of North Carolina at Chapel Hill have been responsible for the reprogramming. The new indexing program, the entry and exit to the IBM package sort using variable length records, and the SUFUN and SUFFIX programs are all operational. THESAUR, the last phase of the VIA program, has been coded but not tested. Plans for the MAPTEXT program will be described in Part III of this section.

The index program has been redesigned so as to cope with texts in prose, poetry, and dramatic forms, as well as with transcriptions of oral discourse. The specifications for the program, as well as the indexing formats, are described in detail in the section of this report devoted to Program Documentation. Our goal has been to write an indexing program with the flexibility to meet a wide range of research needs. Our own research has necessitated the ability to index all the written forms of text mentioned above and we envisage the need for indexing transcriptions of meetings, conversations, etc. In addition to our own requirements, we have had requests for the program which, taken together, would demand the

entire range of its current capacities.

Variable length records are output from the PL/1 INDEX program to auxiliary storage. This output procedure is of interest because it means that we need allocate just the space required for a given word and its associated index. In the Philco 2000 FORTRAN version of this program, we were forced to allot an amount of space equal to the longest word for all indexed words, without regard to word length. For example, the word I and the word COMMUNICATION would both require the same storage space. The use of variable length records, as can be seen, saves significant auxiliary storage especially when processing lengthy texts.

The OS/360 utility sort package is used in the VIA package to sequence alphabetically the INDEX program's output. To handle the sort on the variable length control field composed of the varying length word, a standard exit, E15, is taken during the initial sort assignment phase. At this exit the records are padded to provide the sort's requested fixed length control field. On the sort's final merge phase, another exit is taken, E35, again to provide the varying length record for auxiliary storage.

The program that groups words by root has been divided into two parts, SUFUN and SUFFIX. SUFUN takes care of all the preliminaries to the actual examination of the text and grouping of words by root. SUFUN, therefore, sets up the tables of legal suffixes, the table of exceptions, and the table of function words. Function words are words such as OF, AND, A, THE, etc. -- the words whose primary role in a sentence is syntactic rather than semantic. All words in the text, including function words, are indexed. But only non-function words are grouped according to root and, as a rule, only the indices for non-function words are printed out. Therefore, it is in the SUFFIX portion of the VIA package that content words are separated

from function words. Both SUFUN and SUFFIX are described in this report under Program Documentation.

D. Statistical Analysis of Strings

Word-length character strings have been of demonstrable value for the discrimination of the style of one author from that of another. The work of Mosteller and Wallace upon The Federalist papers[7] and of Ellegard upon the Junius letters[8] depended upon patterns of word choice as definitive clues to authors' identities. We, too, have paid some attention to word length strings when distinguishing between the two translations of Soviet Military Strategy[9]. We would like, however, to deal with character strings not defined by word boundaries in an effort to detect other kinds of graphic patterns, and (when possible) their implicit phonological patterns. Our goal is simply to deal with strings beginning with length two up through as many characters as can be managed -- which may coincidentally occur within word boundaries, which may coincide with word boundaries and which may run across word boundaries. Such patterns of character succession (including punctuation and spaces) may not only comprise in themselves discriminants among styles, but they may also ameliorate problems associated with parsing and other structural descriptions of language.

The statistical problems connected with describing and predicting these strings are considerable. Professor Walter L. Smith, of the University

---

[7] Frederick Mosteller and David L. Wallace, Inference and Disputed Authorship: The Federalist, Addison-Wesley Publishing Co., Inc., Reading, Massachusetts, 1964.

[8] Alvar Ellegard, A Statistical Method for Determining Authorship: The Junius Letters, 1769-1772, Goteborg, 1962.

[9] See pages 68 - 71 in the Second Annual Report.

of North Carolina's Statistics Department, has begun to formulate approaches to the task and Joan Bardez and William Hickok have been working on computer programs which will record and produce frequency counts for strings of up through twelve characters (including spaces and punctuation); the texts on which the programs have been tested are approximately 150,000 words in length, with an average word length of just under four characters. Further reports on the programs and statistical procedures used will be made as this basic research progresses.

III. Plans for Further Research and Development of the Computer-Aided Language Analysis Project

In this brief summary, we will pull together references made in other parts of the report to projected development, and we will also describe several proposed undertakings which are not mentioned elsewhere in the report.

A. Research and Development connected with VIA.

Research on thesauri will continue. That research especially directed toward the elimination of the one manual search now used in VIA includes work on indexing bias in existing thesauri and work on methods for computer-assisted enlargement of existing thesauri. Although a full-scale study of indexing bias will await the availability of existing general-purpose thesauri in computer-accessible form, some preliminary work on indexing bias has already been done and can be pursued further. Comparative runs using Webster's Dictionary of Synonyms, Roget's University Thesaurus, and Roget's International Thesaurus have already been used as a way of exploring distinctions among indices. Next, we will try using common terms on different types of texts in order to explore in depth the retrieval net of a given indexing term. Work begun this past year on the exploration of computer-

assisted enlarging of a thesaurus will continue. The development of programs for a visual display console will be part of this effort.

Related thesaurus research includes the search for methods of using a thesaurus for resolution of semantic ambiguities, and exploration of the possibility of constructing a thesaurus from a large computer-accessible dictionary. The reason for attempting to use a thesaurus for the resolution of ambiguity in some of the words included in VIA's output is that a thesaurus is called upon in the last phase of VIA (THESAUR) and will, therefore, already be available on a random-access storage device. Also, we have thought that because a thesaurus is organized on the basis of semantic associations, it might be possible to resolve ambiguities more easily with a thesaurus than, for example, with a conventional dictionary. Some of the leads we might follow are described on pp. 15-18 of this report. We want to explore the possible use of a large dictionary for thesaurus construction, in part because several dictionaries are already available in computer-accessible form and no general-purpose thesaurus is available, and in part because a dictionary tends to be compiled by a considerable number and range of scholars. Thus we would avoid the consistency of bias which might well exist in the indices and structure of thesauri, since their structure is directly attributable to a small number of scholars.

The reprogramming of the basic VIA program in PL/1 should be completed this year. INDEX and the SUFFIX package (SUFUN and SUFFIX) have been checked out and are operational. THESAUR has been preliminarily coded, but more coding is needed as well as complete checkout.

The heart of the ring-structure version of VIA has been coded in PL/1, but not checked out. The input/output procedures have been

designed but not coded. This phase of the ring-structure VIA should be completed within the next few months. Next, its text handling capacity will be increased to include word phrases. The possibility of setting up the ring-structure VIA so that it may be used in an on-line, time-sharing mode with a visual display console will also be explored.

B. Research and Development Connected with MAPTEXT.

We have not yet recoded MAPTEXT in PL/1, in part because we want to redesign the program so as to include a greater variety of graphing options and in part because we want to provide a visual display console mode for the program's operation. An implementation of MAPTEXT in PL/1 should be available by the end of this year.

C. Research and Development Connected with String Analysis.

Exploration of statistical approaches (e.g. Markov chains) to the characterization of free-format strings will continue. The development of computer programs to support the research will include the capacity for the general analysis of transitional probabilities.

PROGRAM DOCUMENTATION

I. Introduction

This section of the report provides documentation in words, flowcharts, and program listings for the operational portions of the VIA package as well as for the ring-structure version of THESAUR. The latter is included because of its prospective interest to researchers who may be using the VIA package. The earlier version of THESAUR, which will also be available in PL/1, is not described here because the PL/1 version has not been tested out and the logic of the program remains as it was described in the First Annual Report and in the subsequent document, Updating of THESAUR. The thoroughness of the documentation represents our effort to make these programs accessible to other research scientists and scholars with a minimum of effort on their part. We have received a great many requests for these programs from people whose preferences in program documentation range from words to program listings. Were it not for the extensive documentation we have been able to provide through these reports, the cost in time to both the researchers working on this project and to those wishing to use the programs, would have been enormous; alternatively, the programs would simply be inaccessible to others.

The documentation for this report of the INDEX, SUFUN, and SUFFIX programs, as well as some program design and coding, has been provided by William Hickok. The initial work on these programs in terms of design and coding of Terry Ruggles, of the System Development Corporation, has also been a major part of this effort and we wish to acknowledge that indebtedness. Although he is not a co-author of the documentation, he is certainly a co-author of the programs described.

II. Documentation for INDEX, SUFUN, and SUFFIX

by
William Hickok

A. User's Section, General Introduction

This section of the document contains user information for three of the stylistic analysis programs. The three programs are: 1) INDEX, which distinguishes and indexes individual words and punctuation in free format text; 2) SUFUN, which builds a historical data set of suffix endings, exception words, and function words to be used in conjunction with 3) SUFFIX, which identifies content words in the text indexed by INDEX and groups words by root, using a procedure based upon possible suffix endings.

The programs have been written in PL/1 and are operational on the IBM/360 computer operating under the Operating System. Each of the above programs will be covered in detail in the following text of this section.

2.1 INDEX Program

The INDEX program produces a data set containing indexed textual words and/or punctuation marks. Each word/punctuation mark may be indexed as to volume, chapter, paragraph, sentence, and word within sentence location.

2.1.1 Textual Input Record Format

Input to the program is via eighty-character input records. Input record format will be described by input positions.

Positions 1 to 71 - Input record positions 1 to 71 are used to contain free format textual data in the form of words and/or punctuation marks. Each word and/or punctuation mark is delineated

by a blank. For example, the phrase

TOM SAID, "GO TO THE STORE".

would be entered onto a data processing card beginning in position one as

TOM SAID , " GO TO THE STORE " .

Position 72 - Position 72 is blank or contains one of the following special characters:

1. Dash (-) signals a continuation of a word between succeeding input records. If a complete word does not end in position 71, enter the dash in position 72, and continue the word beginning in position 1 of the following record. The word counter will be incremented only once.

2. Commercial "at" sign (@) signals a continuation of a sentence between succeeding records. This operator is valid only for type processing POET or MILT. All other types of processing will cause the operator to be ignored. The placing of this operator in position 72 causes sentence counters not to be incremented. This option is necessary because a line of poetry (either in the form for which the only major structural feature is the line or in the form indicated by MILT -- which may be divided into books, or cantos, or something analogous, as well as lines) may well run over onto a second printed line even though the initial line and the run-over comprise just one poetic line.

3. Pound Sign (#) signals the continuation of both the word and sentence between input records. Use of this operator will cause the word counter not to be reset and the sentence counter

will not be incremented. This operator is valid only for type processing POET or MILT.

Positions 73 and 74 - If type processing is MILT the program expects to find a volume number in two-digit format in positions 73 and 74. The index for Paradise Lost, the work for which the MILT option is currently being used, gives the book number in columns 73 and 74 and the line number in columns 75 - 78. For example, 110232 would refer to Book 11, line 232. The line number may be right-justified in columns 75 - 80, if the programmer prefers.

Positions 75 - 80 - These input record positions are for use in sequencing of the input records. Any combination of character/digits may be used providing they form a logical collating sequence. The collating sequence is not program checked.

Under type of processing, OPLA (Out-of-Sequence Play: for a play such the Ginn & Company, Kittredge edition of Hamlet, in which the printed line numbers and the number of lines referenced on the page seem not to correspond) positions 1 - 68 contain free format input data, positions 69 - 72 contain a play line number which is equated to a sentence number, and positions 73 - 80 may be used for sequencing.

2.1.2 Program Control Key Words

The INDEX program recognizes two types of key words; 1) processing key words, those key words specifying how data is to be processed by the program and, 2) textual key words which are used with passages which have been excerpted from longer texts (such as sections of Hume's History of England) to indicate the location of the excerpt within the text as a whole.

2.1.2.1 Processing Key Words

Processing key words are passed to the program through a Job Control Language (JCL) parameter character string or the first input record of the input data set. Position notation is used, i.e., the program expects to find key words at specified locations within the character string. Each processing key word will be discussed according to its position.

Positions 1 - 4 - Type of Processing - These positions contain a code indicating which type of processing is to be performed. Seven type processing key words are allowed. Entry of other than one of the following key words will cause the program to terminate. The processing key words recognized are:

FRST - This key word must be passed to the INDEX program through the JCL parameter field when all key words are to be found on the first record of the input data set.

PROS - This key word should be used for normal textual data. The volume, chapter, paragraph and sentence indexing counters are reset and incremented by textual key words contained in the text. (See Section 2.1.2.1). Indexed words and/or punctuation are written as they are distinguished on the input records. The word continuation symbol is recognized (input record position 72) but the line and word/line continuation symbols are ignored. The latter two symbols are used just with POET and MILT.

PLAY - This key word causes processing in the same manner as PROS, but with the addition of the processing key word, 'STAGE DIRECTION'. Stage directions are processed as though normal text (refer to 2.1.2.2 for STAGE DIRECTION key word

format). When stage directions are processed, they appear contiguously in the output because each stage direction is preceded by an asterisk. This asterisk remains affixed as the first character of the word.

SPOK - This key word causes processing in the same manner as PROS. However, different textual key words are recognized for resetting counters. (Refer to Section 2.1.2.2)

POET - This key word causes printing of the complete input record rather than individual words and/or punctuation as they are recognized in the input record. A check is not made for textual key words. The inclusion of textual key words in the input text will cause them to be indexed in a normal manner. Volume, chapter and paragraph index counters are each to the value one. The sentence counter is incremented each time an input record is read. The word counter is reset each time the sentence counter is incremented. Word, sentence, and word/sentence continuation symbols in input record position 72 are recognized.

MILT - MILT is a special case of POET. General processing is the same except the volume indexing counter is set via data in positions 73 and 74 of the input record.

OPLA - The use of OPLA causes the program to print-out the entire record as with POET. However, the sentence indexing counter is reset from input record positions 69 - 72 and the PLAY textual key words are recognized including stage directions when the processing key word STAGE DIRECTION has been specified.

Positions 6 - 10 - print control.

These positions follow a comma, which occurs after the processing key word and indicate whether or not the indexed words or punctuation marks are to be printed.

PRINT - indicates that textual data is to be printed.

NOPRT - indicates that textual data is not to be printed.

If these positions are blank, no printing is assumed.

Program control information such as index counter reset values and ending summary messages are printed independent of the print control key word.

Positions 12 - 26 - stage directions.

These positions follow a comma after the print control key word. Specifying the key word STAGE DIRECTION will cause the processing of stage directions (and/or punctuation) as though they were textual words. When the processing key word is PLAY or OPLA the omission of this key word will cause stage directions to be ignored.

Examples for use of processing key words:

1) To process Prose, to print textual data as processed, and to introduce the key words through the JCL parameter field, the OS/360 JCL EXEC card would be:

```
//      EXEC XXXX,PARM.GO='PROS,PRINT'
```

(where 'XXXX' indicates catalogued procedure to be executed.)

utilizing the first input record to contain processing key words, the JCL EXEC card would be:

```
//      EXEC XXXX,PARM.GO='FRST'
```

and the first input record would contain, beginning in column 1

```
PROS,PRINT.
```

2) To process a play, not printing textual data, but processing stage directions, information furnished through JCL would be

```
//      EXEC XXXX,PARM.GO='PLAY,NOPRT,STAGE DIRECTION'
```

2.1.2.2 Textual Key Words

Textual key words are of two types:

1) those key words used to reset indexing counters
2) those textual key words used to increment indexing counters.

Reset Key Words

Key words used to reset indexing counters are prefaced by a dollar sign, followed by the counter name that is to be reset, followed by a value that the counter is to be reset to. Only one key word may appear on an input record. Input positions after the reset value are ignored and not processed. If processing key word is PROS, four reset key words are allowed:

$VOLUME followed by a decimal integer, XX, will reset the volume to that value.

$CHAPTER XXX will reset the chapter indexing counter to any value up through 3 decimal digits.

$PARAGRAPH XXX will reset the paragraph indexing counter to any value up through 3 decimal digits.

$SENTENCE XXXXX will reset the sentence indexing counter to any value up through 5 decimal digits.

For processing key word PLAY or OPLA, two reset key words are allowed:

$ACT XXX will reset the chapter indexing counter to any value up through 3 decimal digits.

$SCENE XXX will reset the paragraph indexing counter to any value up through 3 decimal digits.

The volume indexing counter will equal the value one.

If the processing key word is SPOK, for spoken word, three reset key words are allowed:

$SERIES XX will reset the volume indexing counter to any value up through two decimal digits.

$SESSION XXX will reset the chapter indexing counter to any value up through three decimal digits.

$SPEAKER XXX will reset the paragraph indexing counter to any value up through three decimal digits.

For processing key words other than the four specified above, the textual key words to reset indexing counters are ignored. If textual key words appear in the text they will be treated as normal textual data and be indexed.

Increment Key Words

If the processing key word is PROS, the normal end-of-sentence punctuation, e.g. period, question mark, quotation mark, will increment the sentence indexing counter. When the sentence indexing counter is incremented, the word indexing counter is reset to one. If the processing key word is PROS, PLAY, or SPOK a double period will increment the paragraph counter by one, reset the sentence indexing counter to one, and reset the word indexing counter to one. Three dollar signs in a row within the text will increment the chapter indexing counter by one, reset the paragraph indexing counter, sentence indexing counter and the word indexing counter to one. Similarly, four dollar signs in a row will increment the volume indexing counter by one, reset chapter, paragraph, sentence, and word indexing counters to one. In all cases the double period, three dollar sign , and four dollar sign notation will be changed to a single period before outputting as text.

2.1.3 Output Record Format

Output from the indexing program is variable length, 26-84 character records. Each record contains one word and/or punctuation mark and its associated indexing information. Record format is as follows:

Positions 1 - 4 - Four-byte variable count field. This field contains in binary format, the length of the entire variable length record.

Positions 5 and 6 - Two-byte length of the indexed word.

Positions 7 - 9 - Two-byte Volume number.

Positions 10 - 12 - Three-byte Chapter number.

Positions 13 - 15 - Three-byte Paragraph number.

Positions 16 - 20 - Five-byte Sentence number.

Positions 21 - 25 - Two -byte Word number within sentence.

Positions 26 - 84 - Variable length word.

Output of variable length records is accomplished by using the PL/1 PUT STRING function which converts the indexing counters from internal packed decimal format to character-string format. A call to the subroutine PUTOUT, written in assembly language, computes the length of the entire character format record including the four-byte variable length count field and outputs the record.

2.2 SUFUN Program

The SUFUN program produces a data set containing linked suffix pairs, exception words associated with a suffix pair, and function words. Function words are defined as those words and/or punctuation marks which are not to be considered as content words.

2.2.1 Suffix pair, exception word, and function word input format

The program expects to find all suffix pairs and associated exception

words prior to function words. Each input record contains one suffix pair. The suffixes of the suffix pair must be separated by at least one blank. The first suffix of the pair must begin in input position one (e.g. card column 1). When the first suffix of the pair is a blank, input position one must be blank with the second suffix beginning in or after input position two. The maximum length for each suffix of the pair is eight characters.

Exception words associated with the suffix pair are on the input records (e.g. IBM card) immediately following the suffix pair record. One exception word is allowed per input record. An exception word is denoted by the word EXCEPT beginning in input position one, followed by a blank, followed by the exception word. The maximum length of an exception word is eighteen characters. The number of exception words associated with a suffix pair may vary from one pair to the next; however, the total number of exception words that may be introduced is one-thousand.

An exception to the exception list does exist and is known as the 'Letter Rule'. The letter rule has been introduced to speed processing within the SUFFIX program. The letter rule says that if the final letter of the two words being examined for possible common root does not match the letter introduced as an exception word, the program assumes there is not a legal suffix regardless of the suffix pair and does not look at any further exception words associated with the suffix pair.

The exception word is introduced by the word EXCEPT in input position one of the input record, followed by a blank, followed by the word LETTER, followed by the letter in question. For example: EXCEPT LETTER R means that if the final matching letter in the two words being examined is an R, the suffix pair is legal, unless an exception is found.

Suffix pairs and associated exception words may be introduced to the SUFUN program in any order. The program sorts and links words, suffixes, and exception words before producing the output data set.

In the present program configuration, function words must follow after all suffix pairs and associated exception words. The input record immediately preceding the first function word input record must contain as a single character string 'FUNCTIONWORDS' beginning in input position one. Succeeding input records contain one function word per record. Each word may be up to eighteen characters in length. A maximum of two-hundred function words is presently allowed. Words may be in any order and are sorted before writing to the output data set.

2.2.2 Output Record Format

Six record formats are present in the output data set. All records are twenty characters in length. Figure 1 is a pictorial representation of the Data Set layout showing the order in which the record formats occur in the data set.

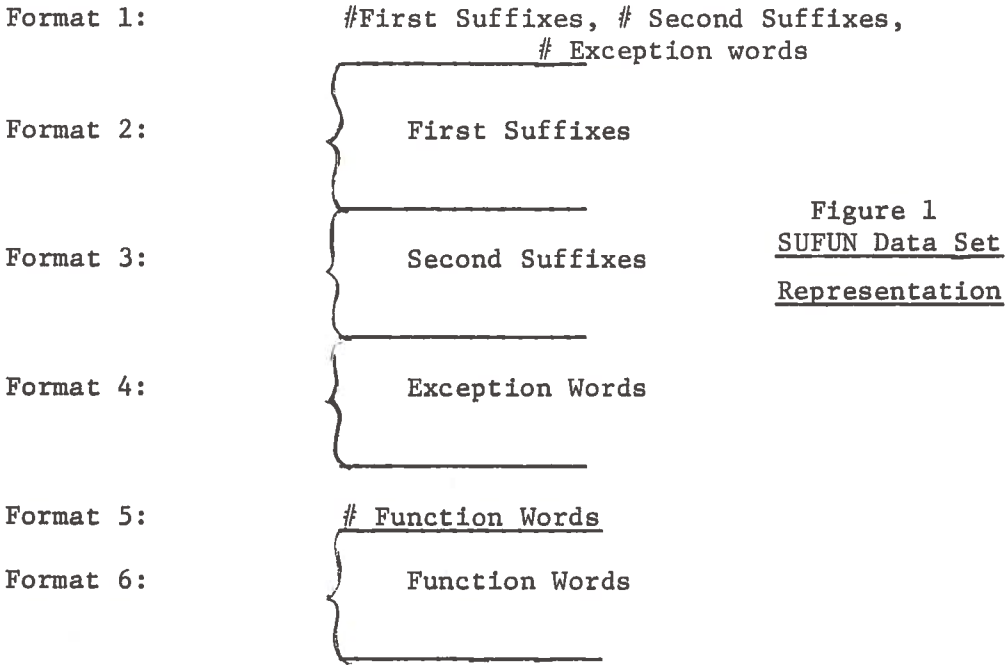


Figure 1
SUFUN Data Set
Representation

1. Number of unique Suffix 1, Suffix 2 and Exception words.
Format:

Positions 1 - 3: number of unique suffix 1's of suffix pair.

Positions 4 - 6: number of unique suffix 2's of suffix pair.

Positions 7 - 10: number of exception words.

Positions 11 - 20: blank.

2. Unique first suffixes of suffix pair. Number of records equals count found in Positions 1 - 3 of record type 1.
Format:

Position 1: length of suffix.

Positions 2 - 4: index to second suffix of pair.

Positions 5 - 12: first suffix, varying in length.

Positions 6 - 20: varying blank.

3. Second suffix of suffix pair. Number of suffixes equals count contained in Positions 4 - 6 of record type 1.
Format:

Position 1: length of suffix

Positions 2 - 5: location of first exception word in exception list associated with suffix pair. Contains 'Ø' if no exception words are associated.

Positions 6 - 13: second suffix, varying in length.

Positions 7 - 20: blanks, varying in number.

4. Exception words. Number equals count contained in Positions 7 - 10 of record type 1.
Format:

Positions 1 - 2: exception word length.

Positions 3 - 20: exception word, varying in length

Positions 4 - 20: blanks, varying in number.

5. Number of function words.
Format:

Positions 1 - 3: number of function words.

Positions 4 - 20: blank.

6. Function words. Number present equals count found in Position 1 - 3 of record type 5.
Format:

Positions 1 - 2: length of function word.

Positions 3 - 20: function word varying in length.

Positions 4 - 20: blank, varying in length.

2.2.3 Printed Output

The program prints the sorted first suffix and its associated suffix and the sorted exception words associated with the suffix pair. Next it prints function words sorted in ascending alphabetical collating sequence.

2.3 SUFFIX Program

The SUFFIX program compares indexed words produced in the INDEX program to function words prepared by the SUFUN program, eliminating indexed words considered function words. The remaining words are considered content words. Succeeding content words are compared character by character under the assumption that at the point of deviation the remaining characters are a possible suffix pair. This possible suffix pair is then used as entry into suffix tables and if a match is found and the word is not an exception word the two content words are considered to have a common root. Content words with common roots are assigned an equal match identification number.

2.3.1 Suffix pair, exception words and function word input record format

Refer to Section 2.2.2 for input record formats.

2.3.2 Indexed word and/or punctuation marks input format

Refer to Section 2.1.2 for the format of the indexed word input record. The indexed words are sorted before introduction to the SUFFIX program.

2.3.3 Output Record Format

Output for the SUFFIX program is variable length, 17 to 74 character records. Each record contains one unique content word and its associated frequency of occurrence and match count. Record format is as follows:

Positions 1 - 4 - four bytes containing the variable count field. This field contains in binary format, the length of the entire variable length record.

Positions 5 and 6 - two bytes containing the length of the content word.

Positions 7 - 11 - five bytes containing the match count linking words with possible common roots.

Positions 12 - 16 - five bytes containing the frequency count for the number of occurrences of the content word in the processed text.

Positions 17 - 74 - these positions contain the varying length content word. A content word may be up to 58 characters in length.

B. Program Documentation

This portion of the report details program operation: A general explanation of the VIA package appeared in Section A. Here we expand on that explanation, providing descriptive material for each of the three programs previously discussed and their major subroutines. To help the user in understanding these programs, a flowchart for each is included. For those interested in greater detail, program listings are provided.

3.1 INDEX Program

For a graphic representation of the INDEX program information flow, refer to Figures 3.1 - 3.4. These flowcharts present a comprehensive explanation of the program.

The program first initializes all indexing counters and opens the data sets used by the program before processing keywords passed through JCL or the first input record of the input data set. An error in specification of processing keywords will cause the program to terminate. After setting indexing counters whose initial values are dependent upon the type of processing keyword, the subroutine, FORM, is called. The subroutine FORM reads an input record and separates out textual words and/or punctuation marks from free format text using blanks as word delimiters. That is, characters between blanks comprise an indexable word.

On entry into FORM, the word length counter NUMC is set to zero and the field which will contain the word is set to blank. The input record position pointer is tested for a value equal to 72 and if equal, input record position 72 is tested for the word, sentence, or word/sentence continuation symbols. If one of these three symbols is found, the appropriate processing "switch"[10] is set on. If the position pointer is greater than 72, a new input record is read. When the processing key word contained in the field UHLAB is equal to the value POET or MILT the word and sentence continuation processing switches are tested. If they are on, the word and/or sentence indexing

---

[10] A processing switch is a field in the program whose value is set to Ø or 1 to denote an on or off condition. The field value may be tested by subsequent program statements, thus acting as a switch.

counters are not incremented. If these processing switches are set on and UHLAB value equals SPOK, the word and/or sentence indexing counters are not incremented.

The subroutine proceeds to scan the input record. A blank in an input position will cause a RETURN to the calling program with all input positions prior to the blank concatenated to form a textual word. Upon return to the main processing routine, the textual word is tested for equality with any of the textual key words which results in resetting any or all indexing counters. If the word is a key word, a branch is taken to the appropriate indexing counter reset routines via the use of the PL/1 variable table capability.[11]

The textual word is written to an output data set using the subroutine PUTOUT (refer to Figure 3.4) This subroutine is written in 360 Operating System assembly language. When called, it is passed the address of the character string containing the converted indexing counters and indexed word. The subroutine opens the output data set the first time it is called. It computes the length of the variable length record using the constant value of 18 for the length of the indexing portion of the record and the value contained in NUMC for the length of the indexed word. The constant CNST is passed as the first two bytes of the passed character string. On succeeding calls, the PUTOUT routine writes a record containing an indexed word. This subroutine closes the output data set when called via CALL CLSEOUT.

---

[11] When the textual key word is an incrementing key word, its value is changed to a single period before the values in the indexing counters are converted to character format.

On return from the PUTOUT subroutine, various counters are incremented. Incrementing is dependent upon the type of processing specified and the textual key word encountered as a textual word.

The program branches back to the label, L1, when NOPRT has been specified as a processing key word and another textual word is separated out by the subroutine FORM. When PRINT is specified, the program prints the indexed word without indexing information before branching to L1. In the case of MILT, POET and SPOK type processing, the entire input record is printed when read by the subroutine FORM and when PRINT is specified.

Summary information is printed when an END-OF-FILE condition is encountered on the input data set. This information is printed regardless of the processing key words; the program then proceeds to close the input and output data sets before terminating.

3.2 SUFUN Program

The function of the SUFUN program is to link the first and second suffixes of a suffix pair with associated exception words, and to process function words; this information is used as input to the SUFFIX program. To conserve internal and external storage facilities, SUFUN eliminates duplicate first suffixes while maintaining the proper relationships to second suffixes and associated exception words. SUFUN constructs a table structure of first suffixes, second suffixes, and exception words from input records before proceeding to process function words. After processing suffixes, function words are input and sorted. Figures 3.5 and 3.6 portray the SUFUN program information flow.

SUFUN begins by initializing counters and opening input and output data sets. SYSIN contains the input records, SYSPRINT is the printer, and the output data set is SUF.

A call to the subroutine FORM is issued. FORM obtains an input record from SYSIN and separates out the first suffix ending or the value EXCEPT. When the value returned by FORM is not EXCEPT, the structure SUFFIX1 is set to the first suffix and its length, and another call for FORM is issued to obtain the second suffix of the pair. The second suffix is then placed in SUFFIX2 along with its length. When the initial call results in the value EXCEPT, a second call is issued and a check is made for the value LETTER. When LETTER is present, a third call to FORM is issued and the letter returned is placed in the associated exception word table. When the second call does not result in the return of the value LETTER the value is considered an exception word and placed in the associated exception word table.

When FORM returns the value FUNCTIONWORDS all suffixes have been examined and are now ready for sorting. A branch is taken to SORTSUF, where a comparison is made of the first and second suffix endings and, if necessary, the values are interchanged so that the lowest value in collating sequence is the first suffix.

SUFUN proceeds to sort the first suffixes, maintaining the proper relationship with the second suffixes. Duplicate first suffixes are then eliminated and a link established to all associated second suffixes.

Second suffixes are now sorted so that each group of second suffixes associated with a single SUFFIX1 is arranged in alphabetical order. This sort is followed by a similar sort for exception words.

After sorting is complete, SUFUN writes to the output data set, SUF, an initial record containing counts of the number of first, and second suffixes and exception words. This initial record is followed by a record

for each first suffix in the SUFFIX1 table followed by records for each second suffix contained in the SUFFIX2 table, and finally by a record for each exception word. The suffixes and associated exception words are then printed.

SUFUN next proceeds to process function words. Succeeding calls are made to FORM; each call results in the return of one function word. When END-OF-FILE on the input data set is reached, the function words are sorted. An initial record is written to SUF after the last exception word. This record contains the total number of function words processed. A record for each function word follows this total record. Initial records are used by the SUFFIX program to read records from the SUF data set.

Function words are printed at the same time they are written to the SUF output data set. The program terminates after output of all function words processed and the closing of the input and output data sets.

The program prints error messages when overflow of a structure occurs. After overflow occurs the program terminates with an appropriate correction procedure message.

3.3 SUFFIX Program

The SUFFIX program accepts output from the INDEX and SUFUN program. Utilizing this output as input, the SUFFIX program eliminates function words from INDEX output(the text), counts the number of occurrences of the remaining content words, and determines common root forms by comparing the endings of words, beginning at the point of deviation between the words. The remaining portions of the words are considered possible suffixes. These possible suffixes are used as entries into suffix tables. When a match of word endings to suffixes is successful, the words are considered to have

a common root form unless an exception word is present.

After initialization of count fields, the SUFFIX program opens input data sets and reads into internal structure arrays all first and second suffixes and function words output by SUFUN. The subroutine CONTENTPROC is called, entering the first time at FIRSTWORD. This subroutine reads text words which were output by INDEX and sorted using the OS/360 sort package and compares each record to the table of function words. If the input record is a function word, it is ignored and a new record is read. When the record is not a function word, it is considered a content word. Additional input records are read, maintaining a frequency count of the number of occurrences of the content word, until a non-match record is found. The subroutine then returns to the main program returning the content word and a total of its occurrences within the text. (Refer to Figure 3.7 to 3.9.)

The main procedure compares the first three characters of the returned content words with the previously returned content word. When a match occurs, the content word is placed into the next position of the structure ROOTCK. A call is then made to CONTENTPROC for another content word. The structure ROOTCK contains only content words for which the first three letters are identical. When a match does not occur, the program begins to process the contents of ROOTCK. A check is made for a single content word entry in ROOTCK and if present, it is assigned a unique match count, MATCNT, value and written out to a data set. Output records are variable length and written using the subroutine PUTOUT described in Section 3.1.

If more than one entry is present in ROOTCK, the content word at the beginning of the structure (indexed by M) is compared with the word at the end of the structure (indexed by N). A comparison is made for content words

which do not have a match count, i.e., which have not been paired with another word(s). If the content word (M) has a match count, then a search is made for word (N) which has no match count. When both words (M) and (N) have match counts, (N) is decreased until it reaches a word (N) without a match count and an attempt is made to match it with word (M). If word (N) equals word (M), and has no match count, then the match count of the word indexed by (N) is set equal to the match count of (M). These words are considered to have a common root and match counts of common roots are equal. The process of decrementing (N) and comparison continues until (M) is equal to (N). At this point, if (M) does not equal the total number of entries in ROOTCK, (M) is incremented by one and (N) reset to the last entry. By this process, (M) proceeds forward through ROOTCK until reaching a content word which has no match count. All content words in ROOTCK have been processed when (M) equals the number of entries in ROOTCK. At this point, the content words in ROOTCK are written to a data set using PUTOUT and the content words which caused the non-equal condition in the first three characters is entered as the first element of ROOTCK. The process then repeats with a call to CONTENTPROC for another content word. The program terminates with an end of file condition on the input data set containing indexed words.

The process of comparing succeeding words in ROOTCK (indexed by (M) and (N)) for possible suffixes and common roots is accomplished by the subroutine STEM.

When called, STEM first checks for a final apostrophe ('), single (s), or apostrophe s('s) on the two content words. When present, these final endings are removed and the two words again checked for equality. If equal, the words should obviously be grouped as a match and return is made to the main routine with the same-root identification made. If the words are not equal,

a comparison of the two content words is made, character by character. When the point of deviation is reached, the remaining portion of content word(M) is placed in REMAIN1 and the remaining portion of content word(N) is placed in REMAIN2. REMAIN1 and REMAIN2 contents are compared and switched, if necessary, so that REMAIN1 will contain the lowest value in collating sequence. This is necessary because the tables containing suffixes have been sorted so that the smaller of the suffix pair occurs in the SUFFIX1 table. A check is made to assure that the possible suffixes are less than eight characters in length (the longest suffix perused by SUFUN) and return is made to the main procedure if the remaining possible suffix is greater than eight characters.

A binary search is made in the SUFFIX1 table using the contents of REMAIN1 for the argument. If the suffix is found, an entry is made to the SUFFIX2 table based on the starting address found in the field of LOCSUF2 associated with the first suffix. A binary search is made within the associated second suffixes using the contents of REMAIN2 as an argument. If a match is found in SUFFIX2, a table look up search is made of associated entries in the exception word list. A zero in the field LOCEXC associated with the matching SUFFIX2 entry indicates that there are no exception words, thereby signalling that a legal suffix pair has been found. If there are exceptions, the appropriate list of exception words is searched until a zero is reached. After testing for a zero, a test is made for the LETTER exception. A comparison is made using the last letter of content word M as an argument. If the letters are the same, the LETTER rule applies, the suffix pair is still legal, and a further search of the exception list is made. If the letters are not equal, a further search of the exception list is made, looking for another LETTER rule entry. Only if an exception is found, or if the LETTER

rule exists but there is no match between letters, control will be returned to the main program without a match indication. Refer to Section 2.3 for further explanation of the letter rule. When all entries have been compared using the content words as arguments and no match is found, then a legal suffix pair has been found.

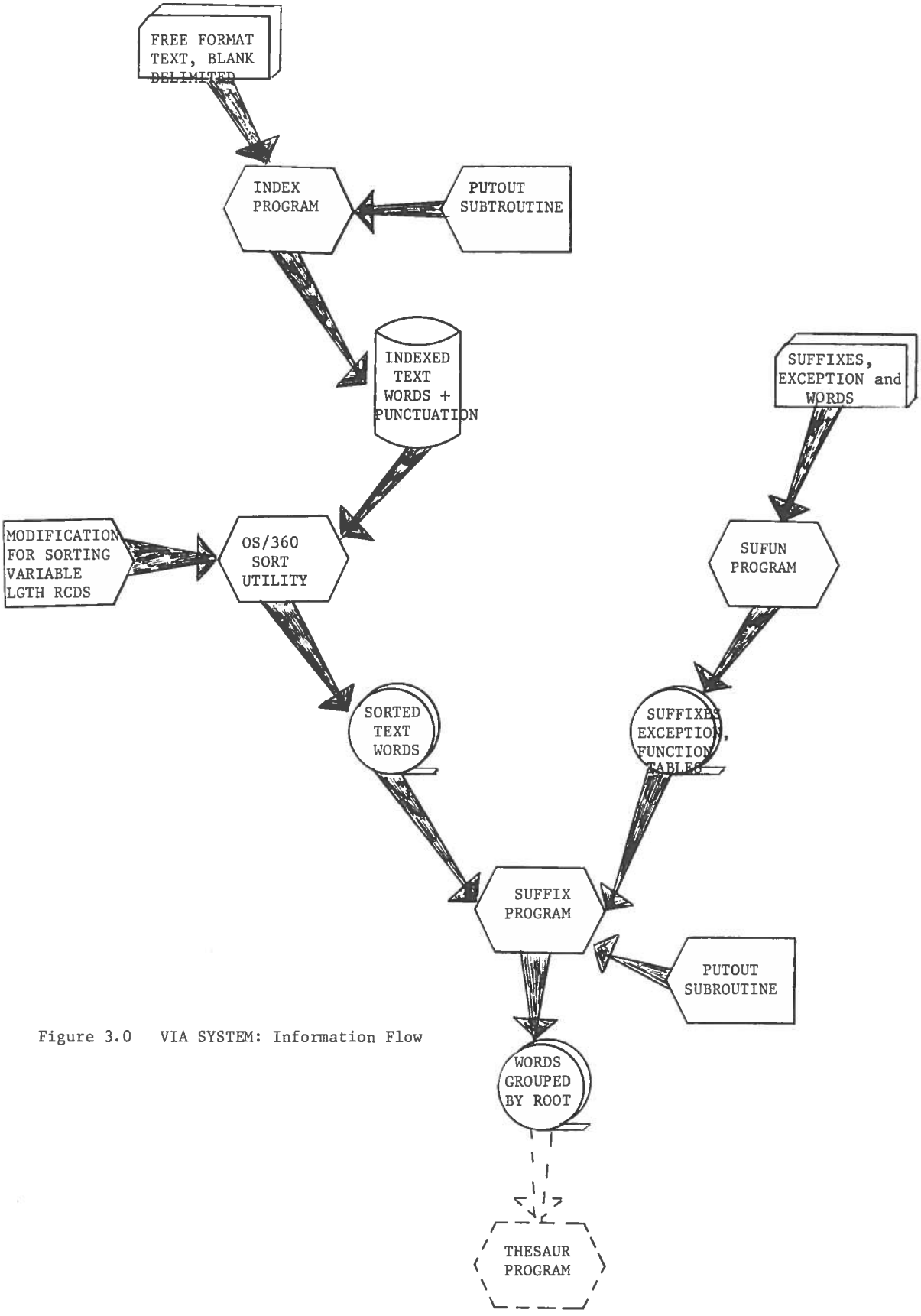


Figure 3.0 VIA SYSTEM: Information Flow

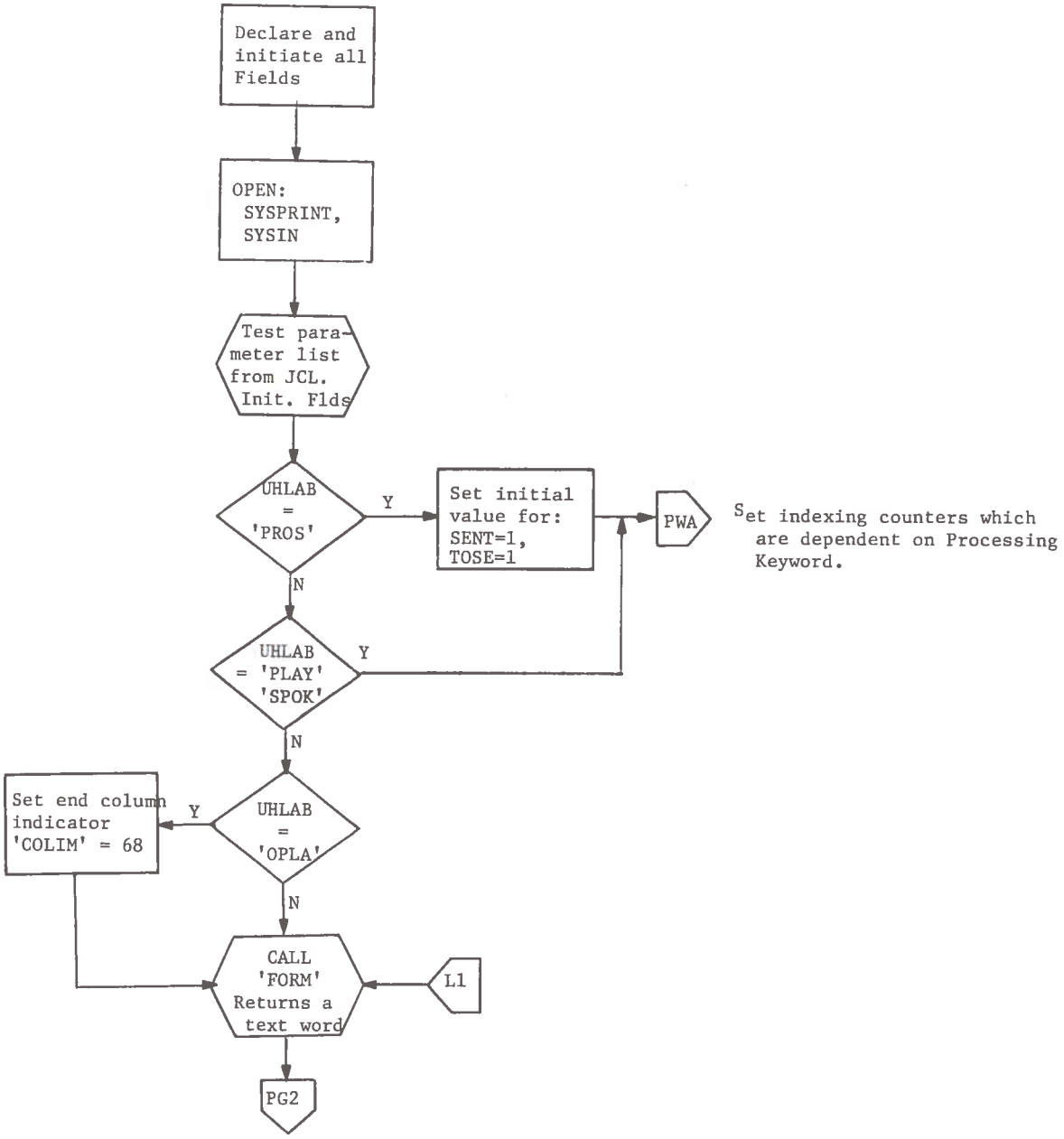


Figure 3.1: INDEX Program: Main Routine

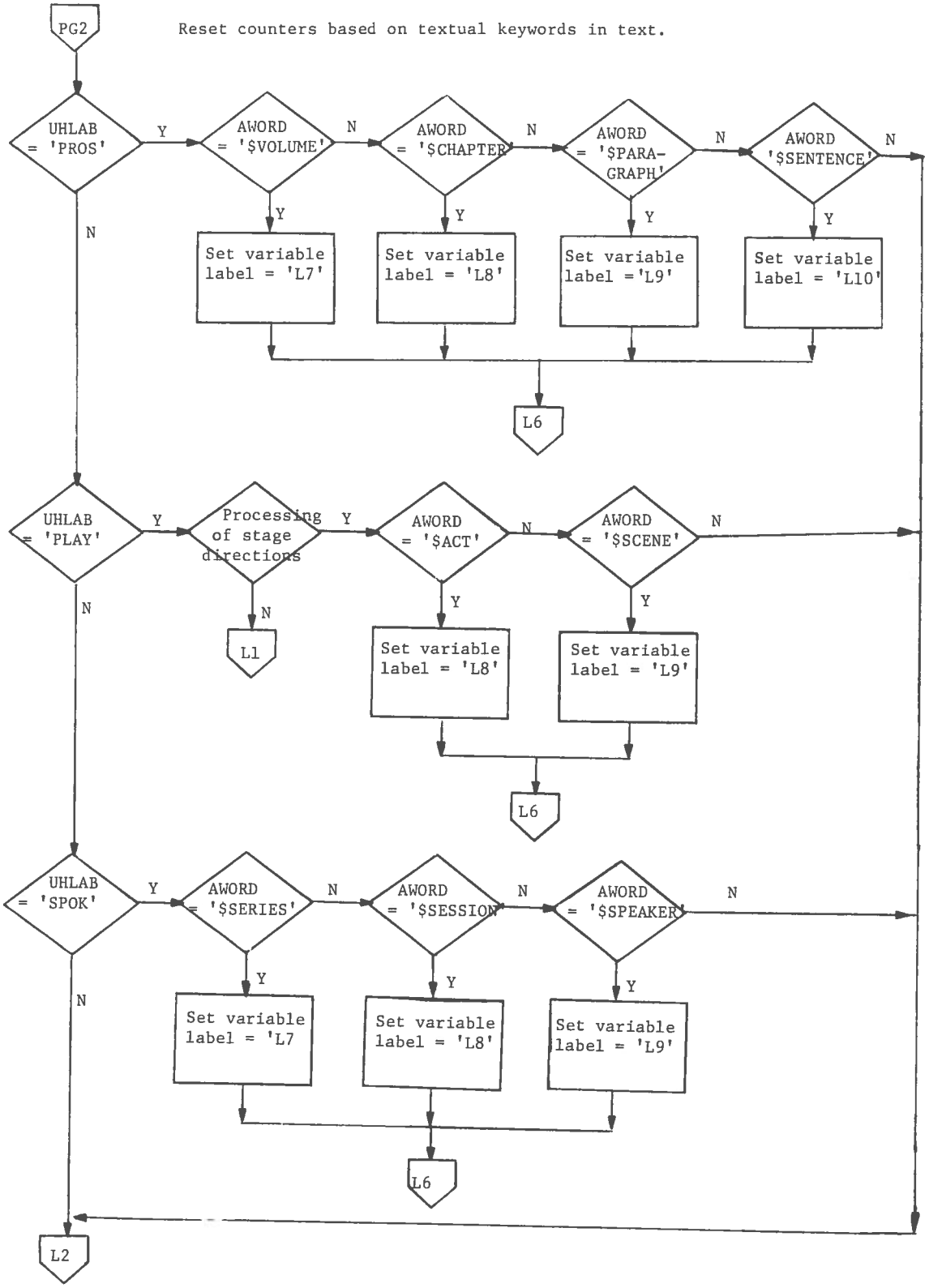


Figure 3.1-2: INDEX Program: Main Routine

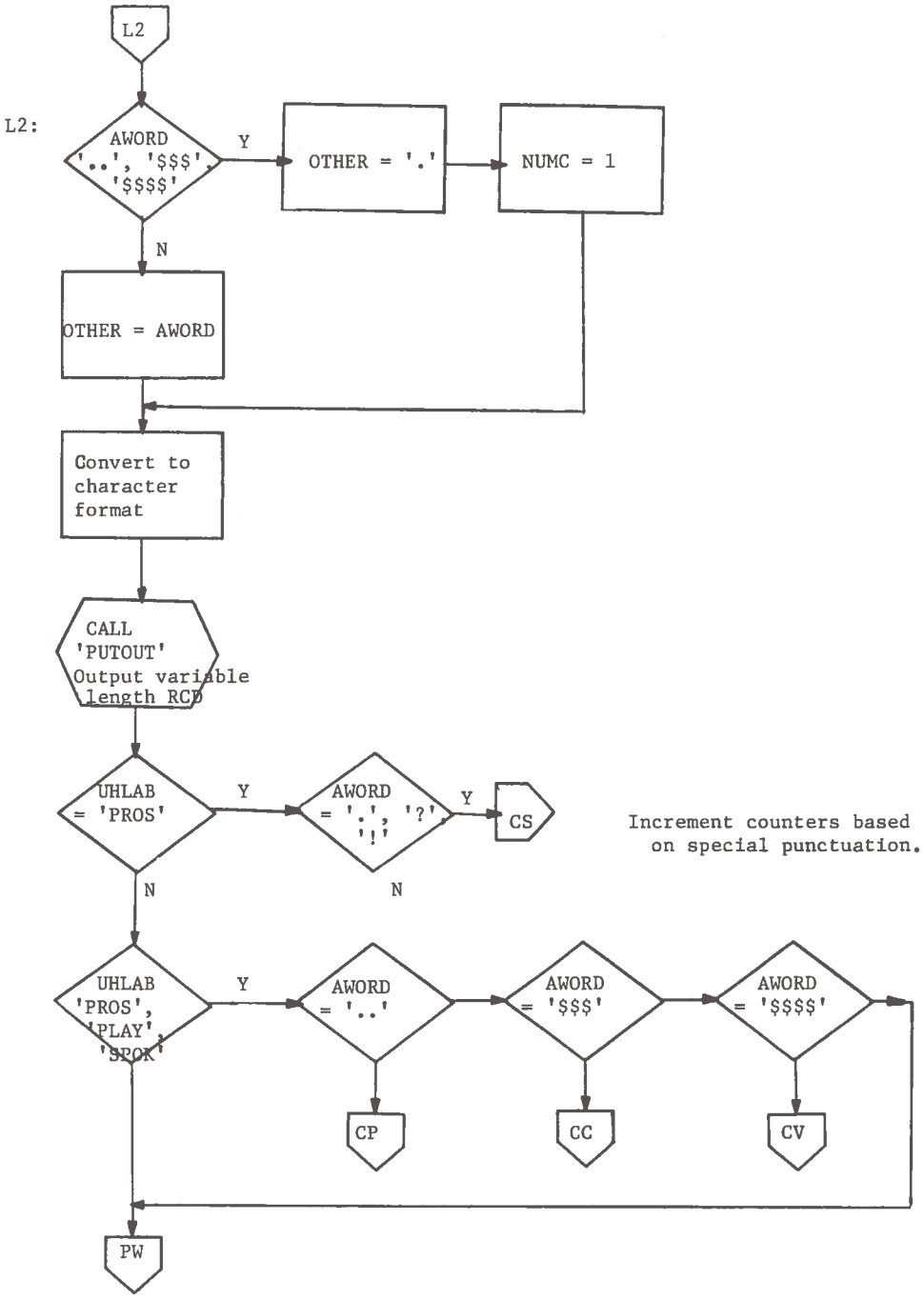


Figure 3.1-3: INDEX Program: Main Routine

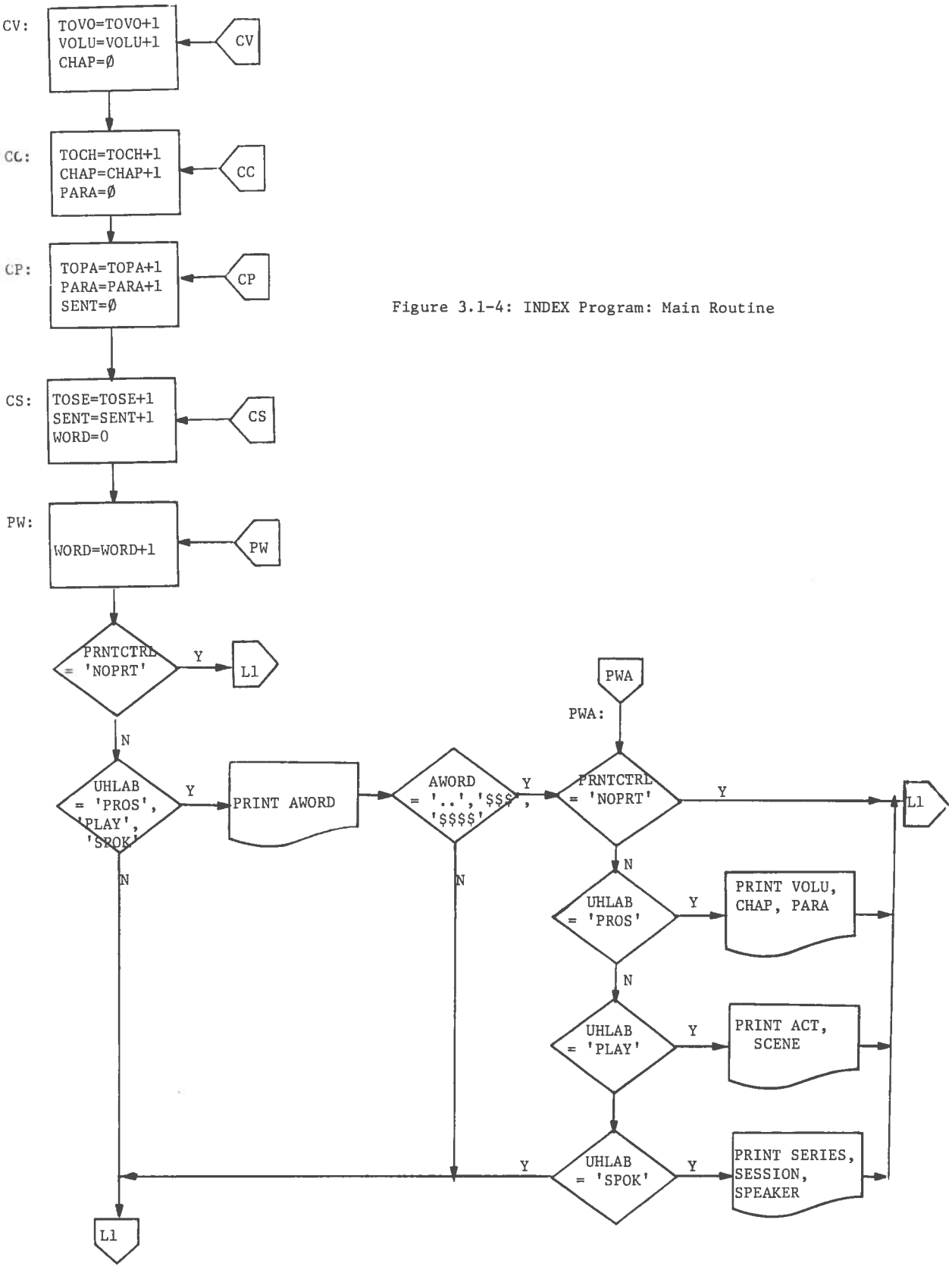


Figure 3.1-4: INDEX Program: Main Routine

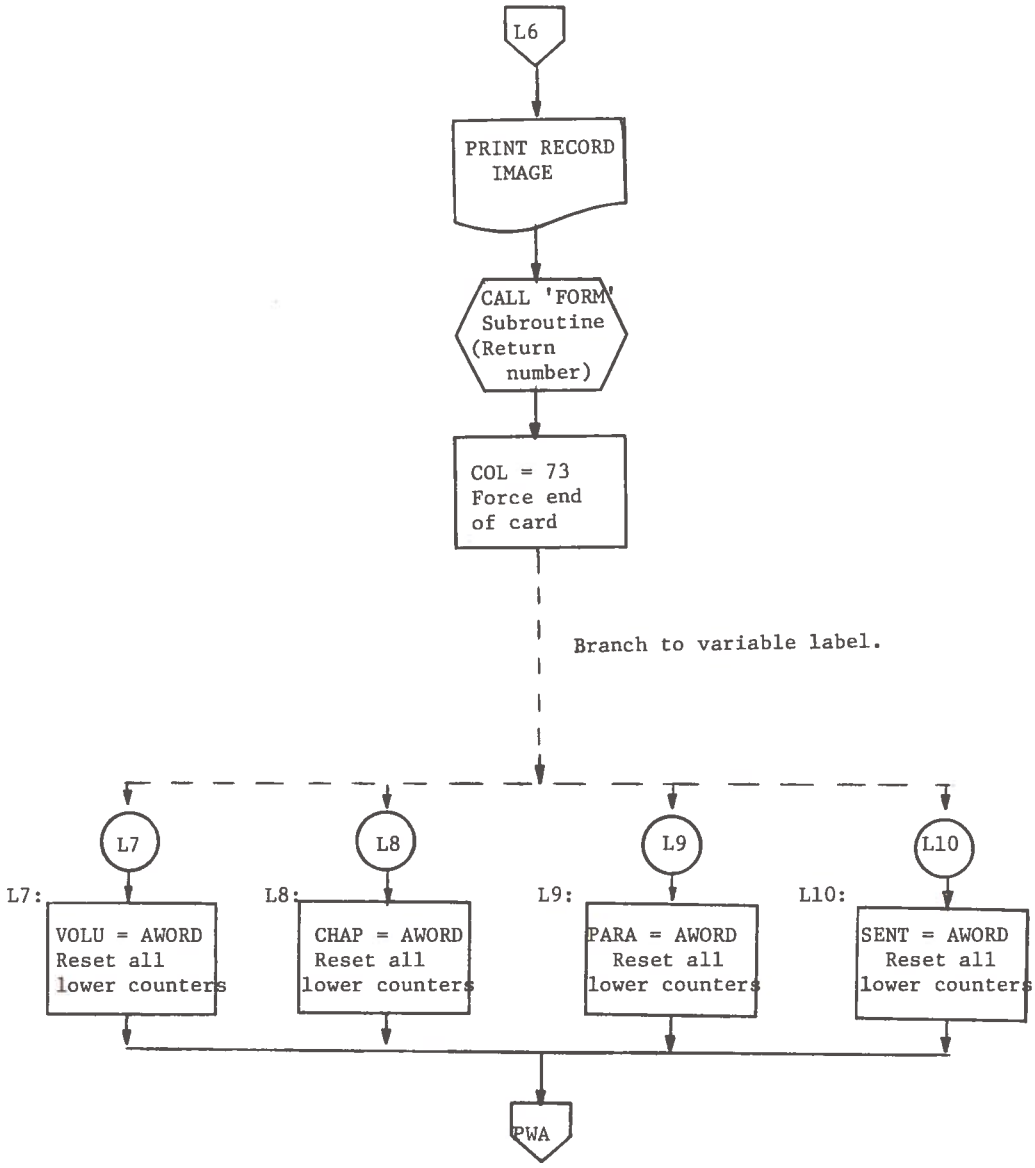


Figure 3.1-5: INDEX Program: Main Routine

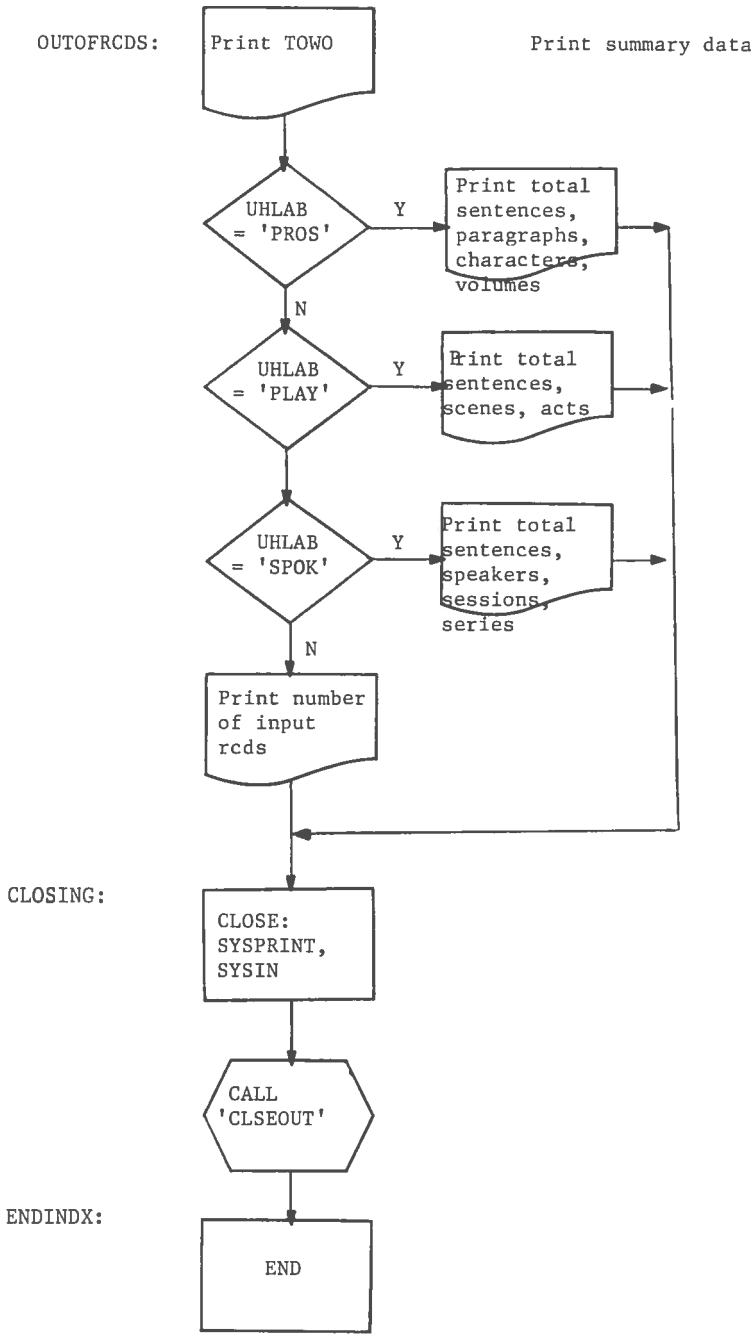


Figure 3.1-6: INDEX Program: Main Routine

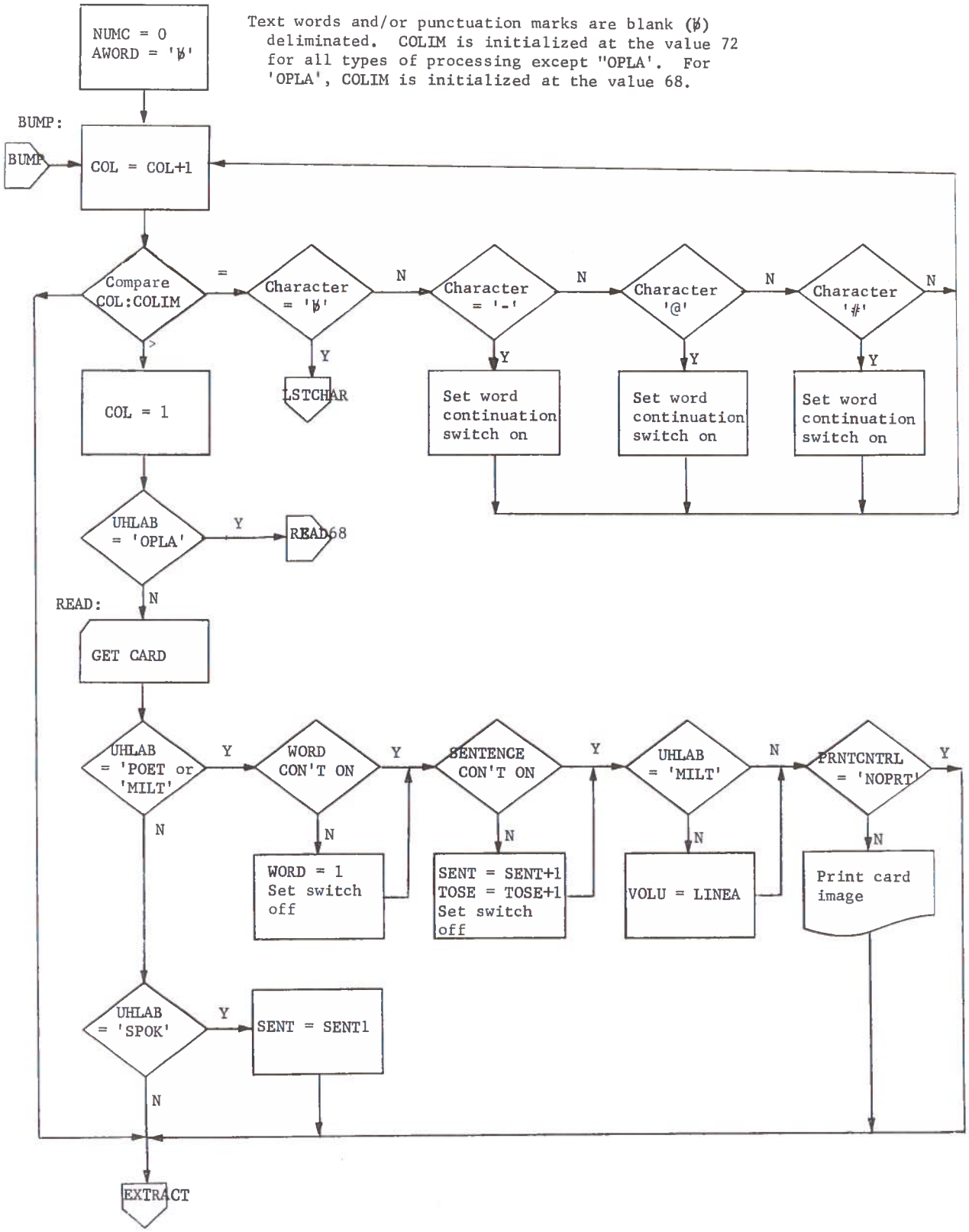


Figure 3.2: INDEX Program: Subroutine FORM

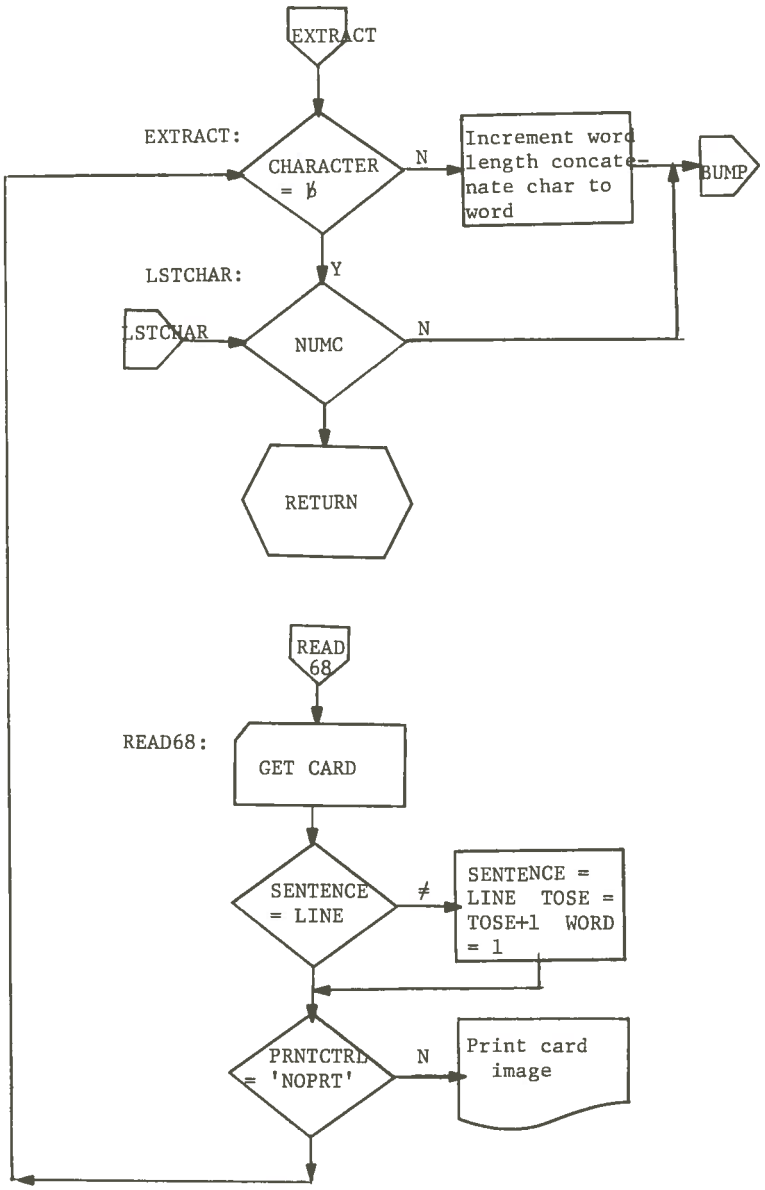


Figure 3.2-2: INDEX Program: Subroutine FORM

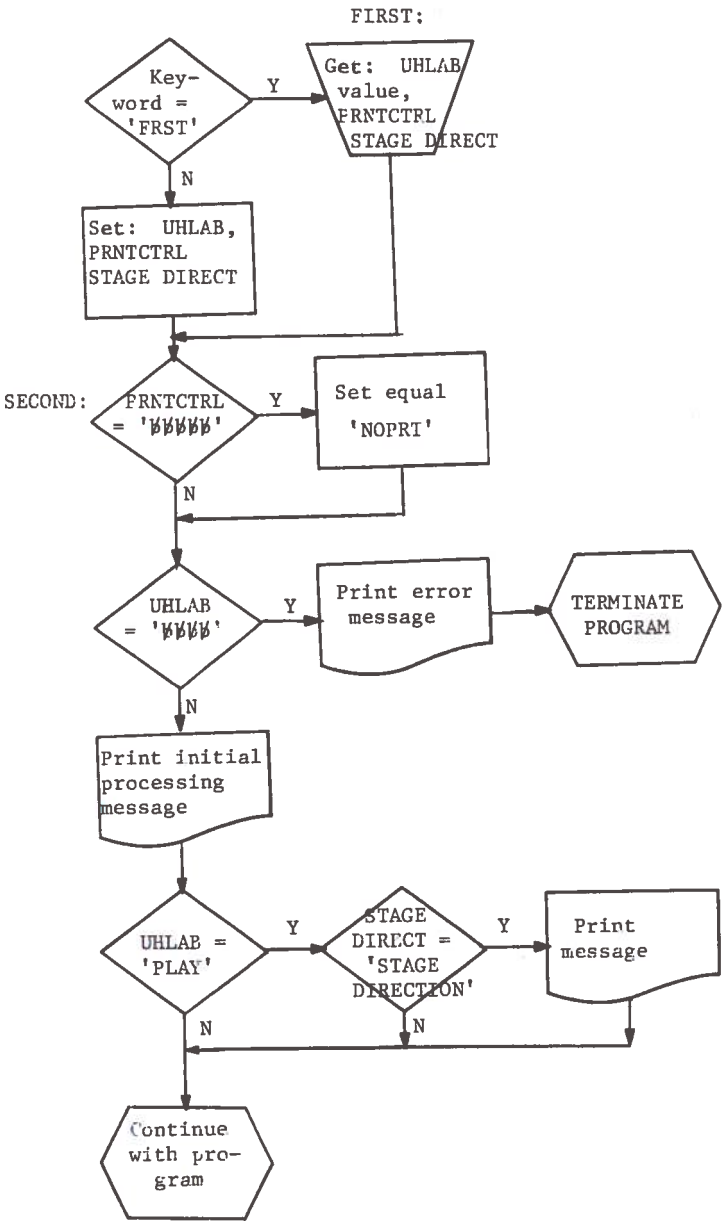


Figure 3.3: TESTING PARAMETER LIST
Passed through JCL EXEC statement.

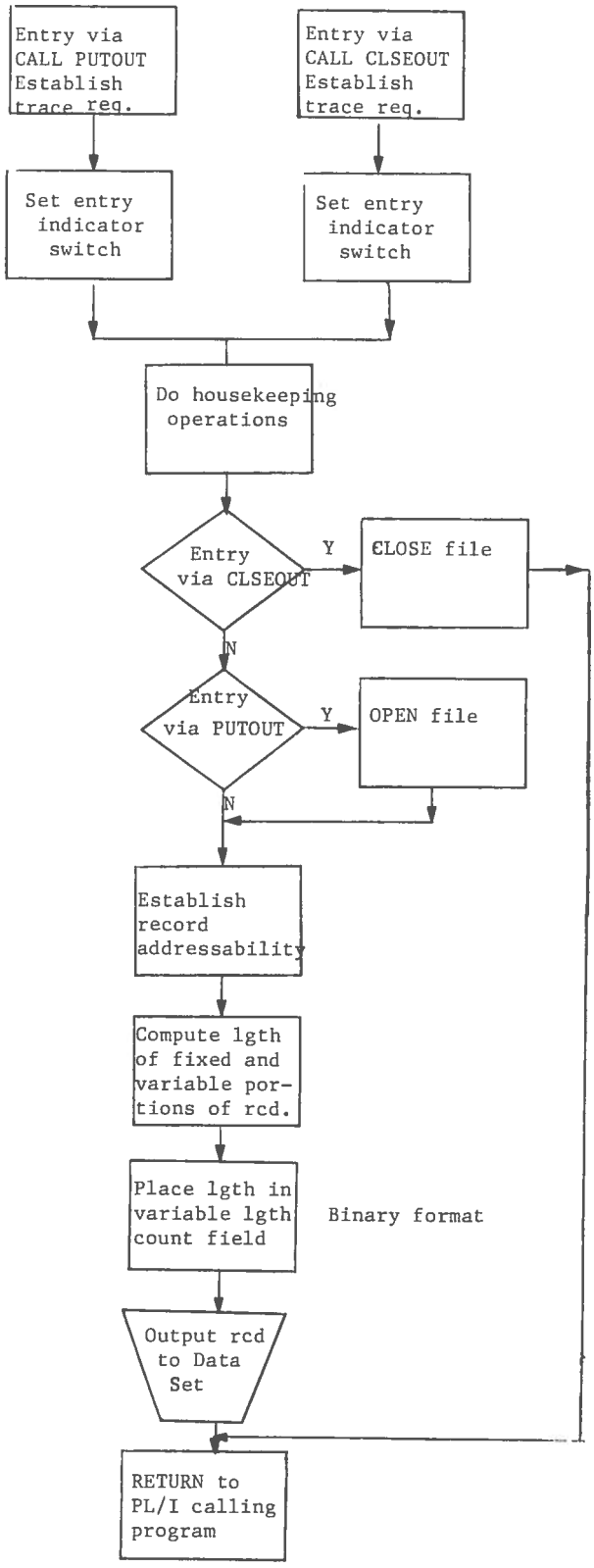


Figure 3.4: PUTOUT Subroutine

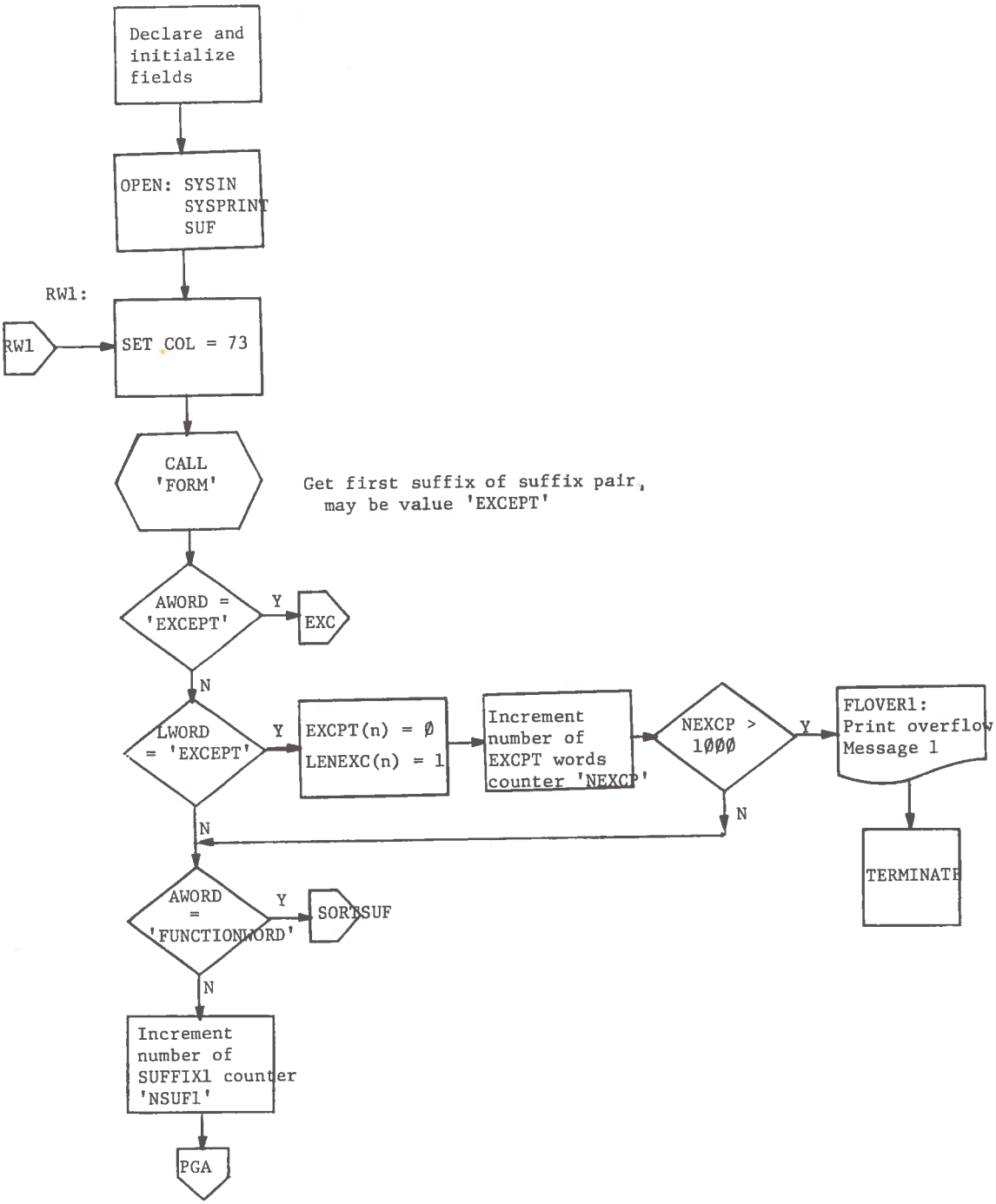


Figure 3.5: SUFUN Program: Main Routine

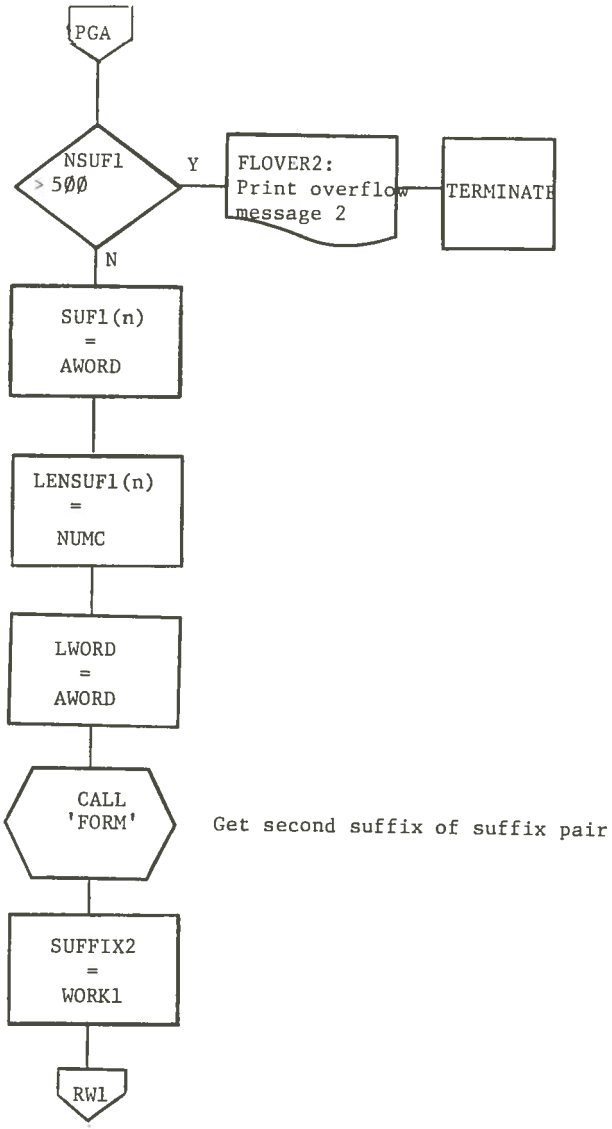


Figure 3.5-2: SUFUN Program: Main Routine

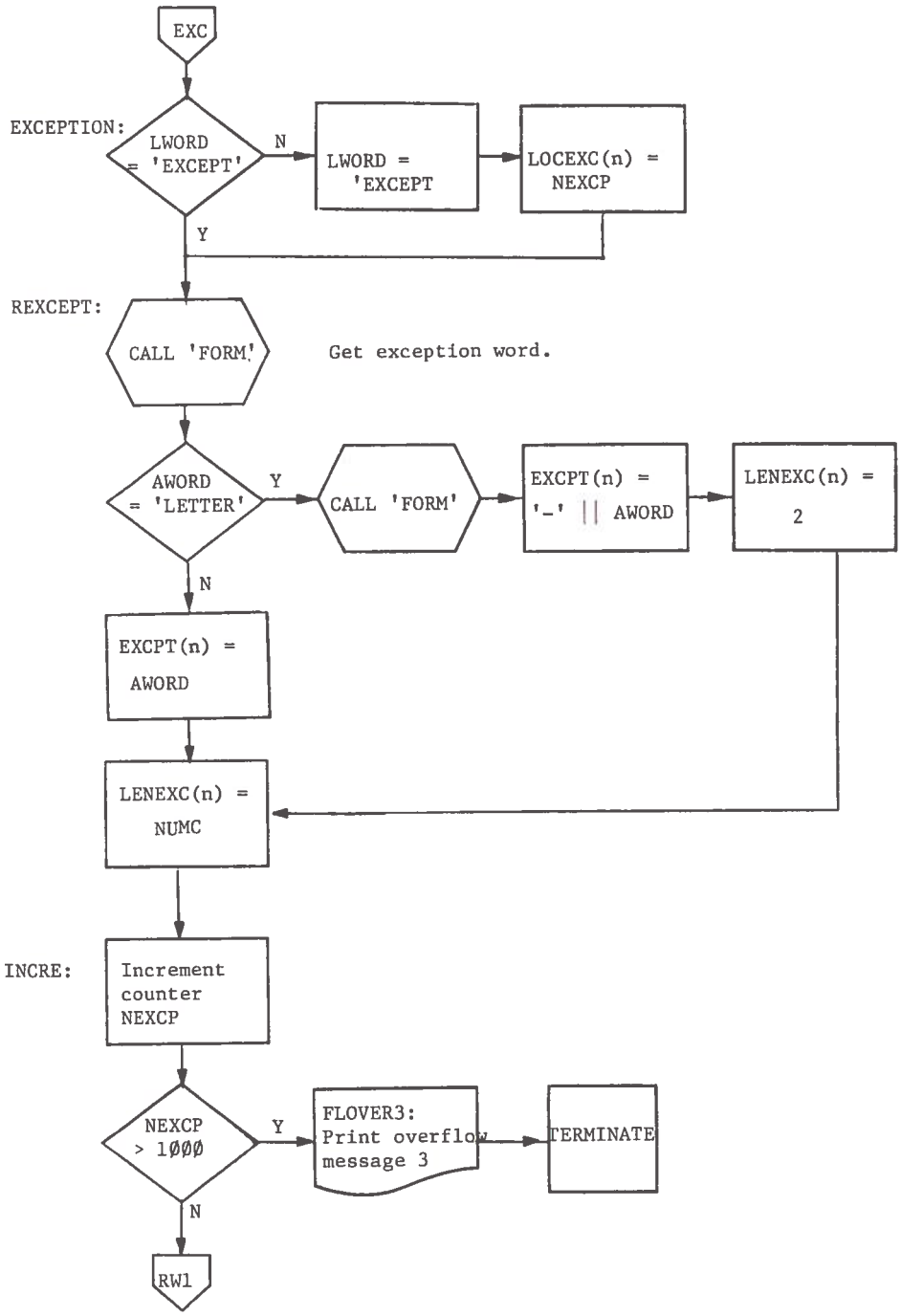


Figure 3.5-3: SUFUN Program: Main Routine

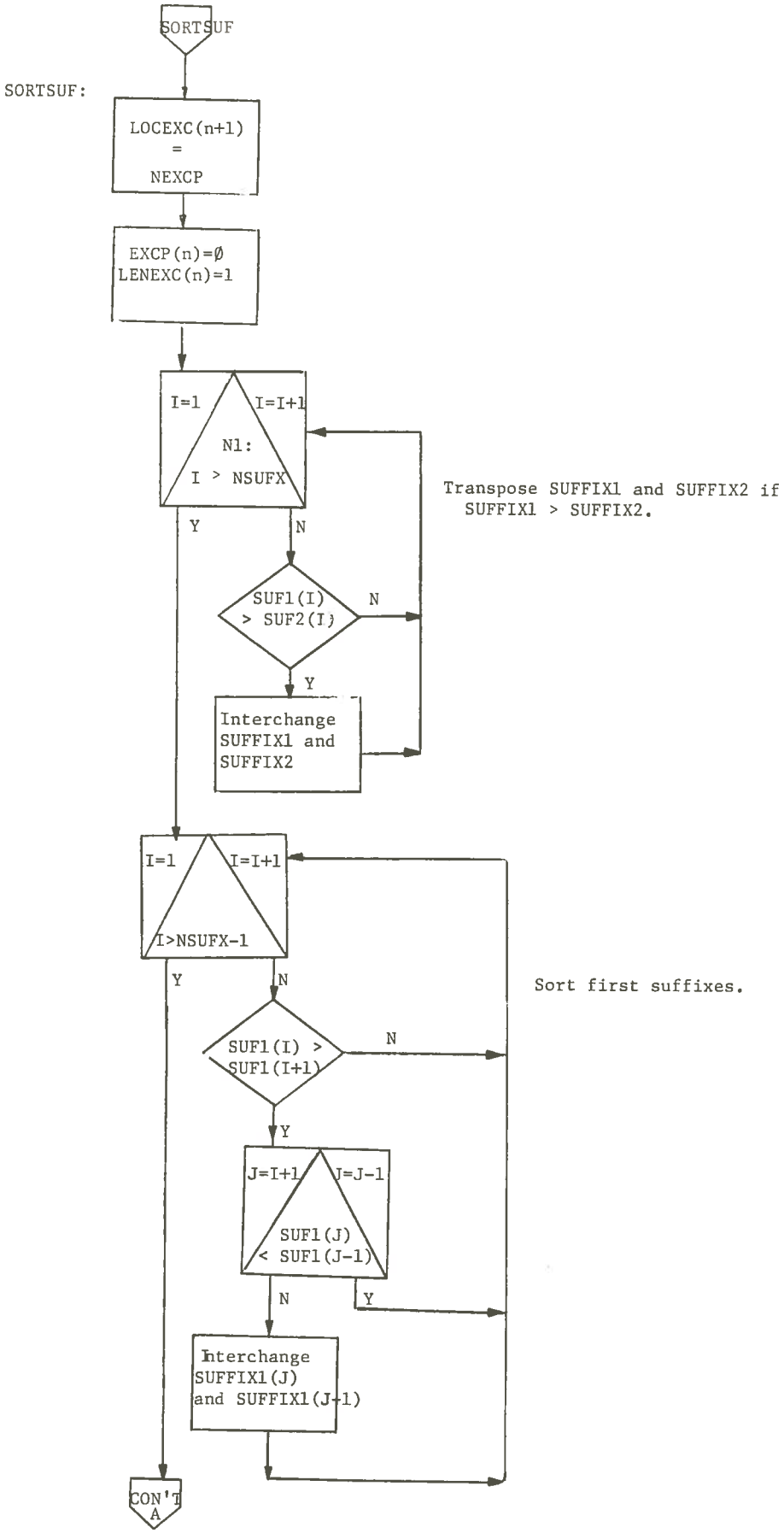


Figure 3.5-4: SUFUN Program: Main Routine

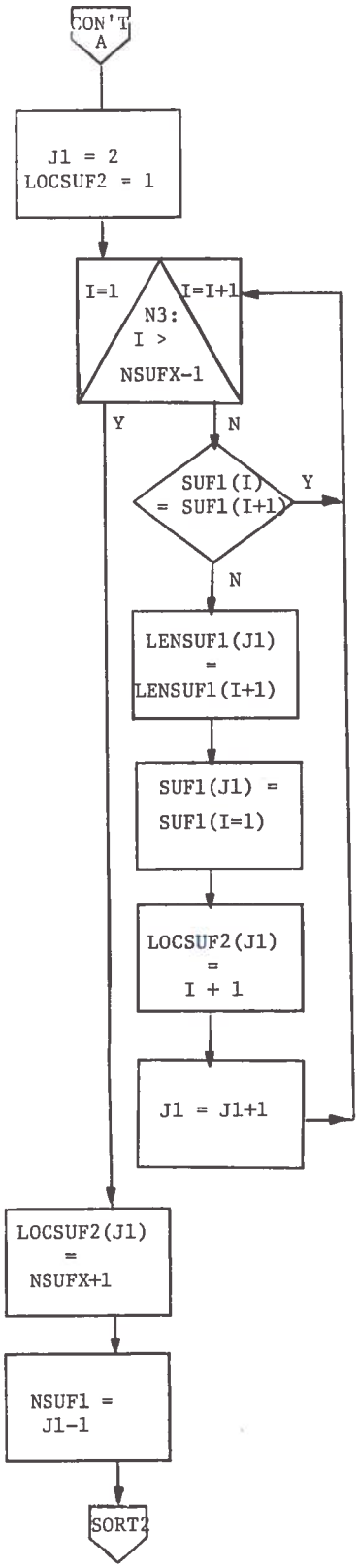


Figure 3.5-5: SUFUN Program: Main Routine

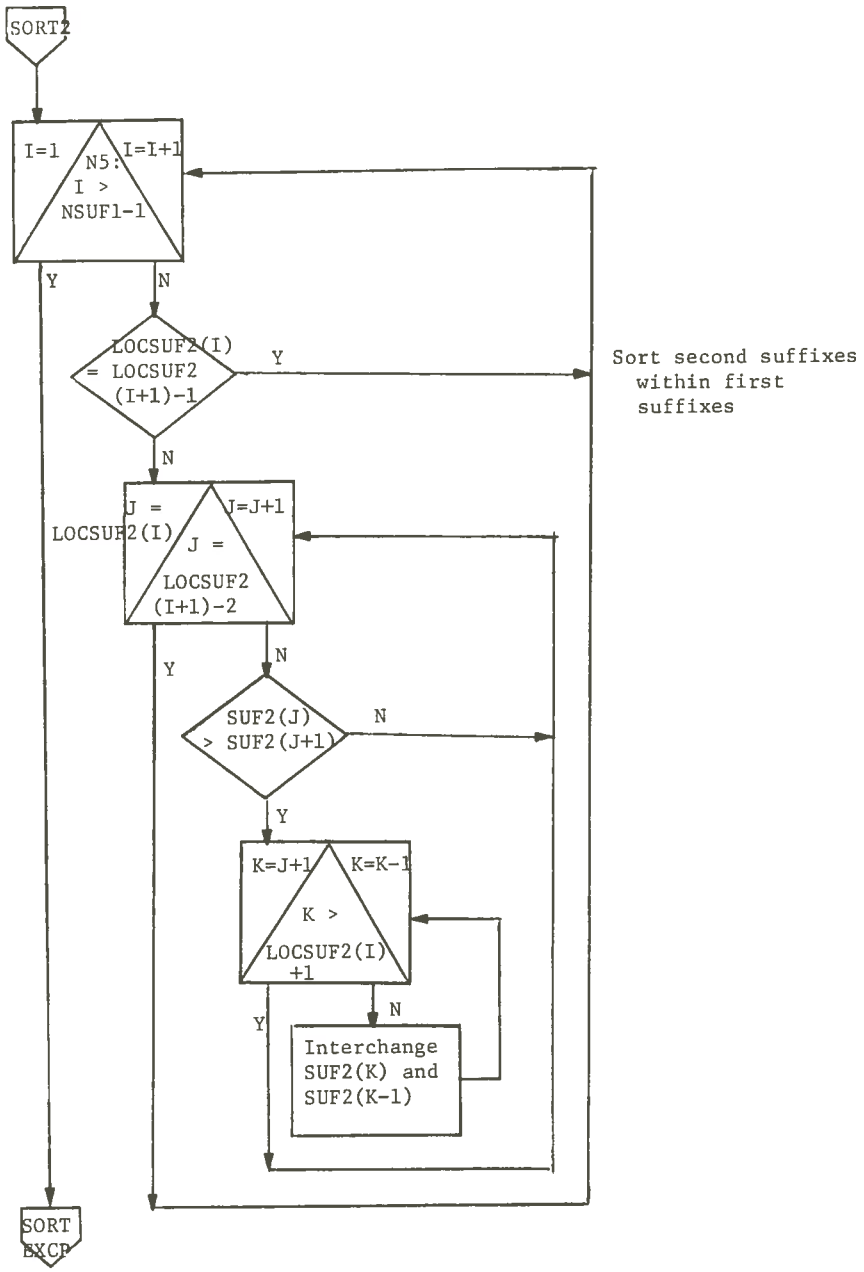


Figure 3.5-6: SUFUN Program: Main Routine

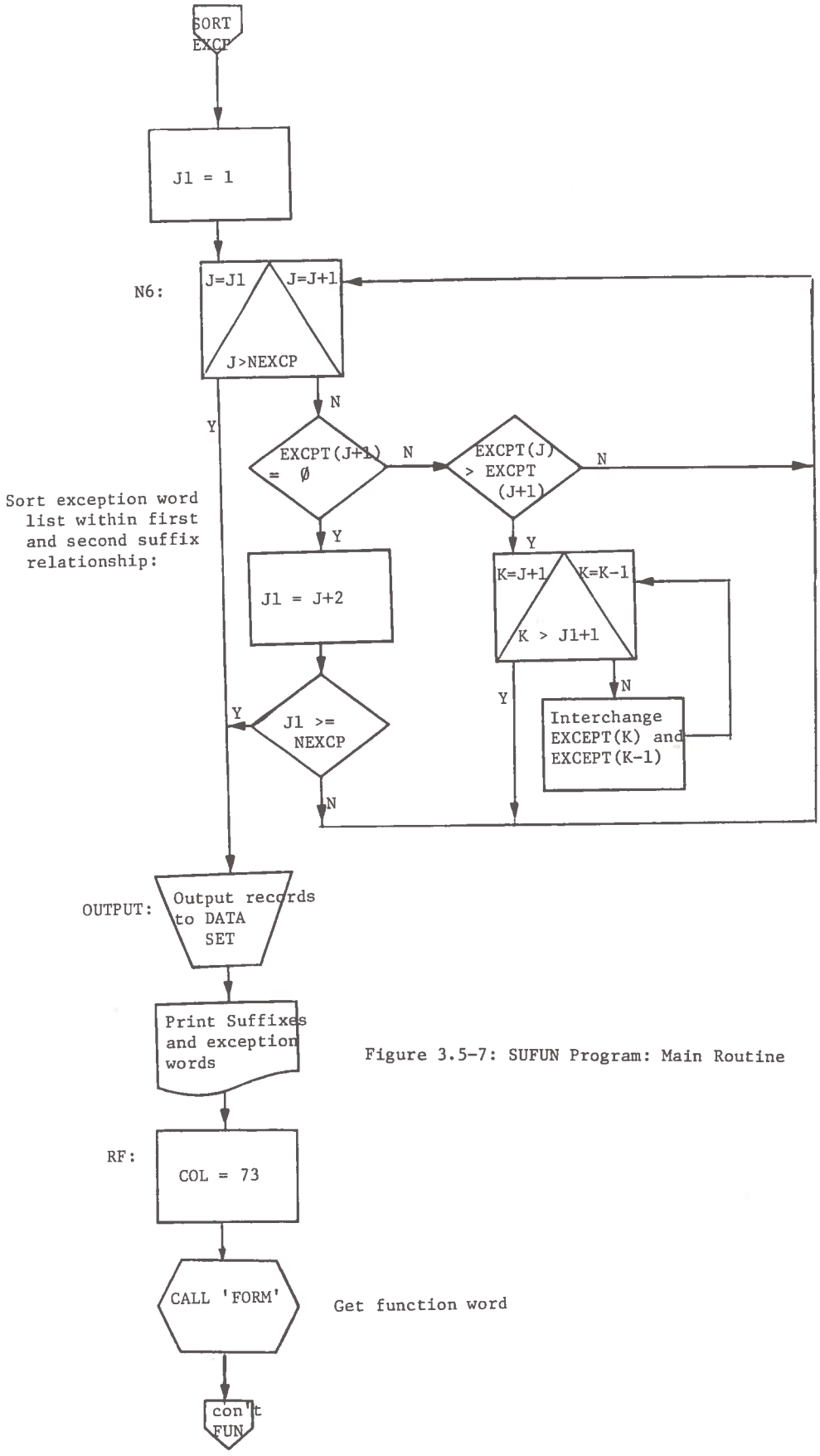


Figure 3.5-7: SUFUN Program: Main Routine

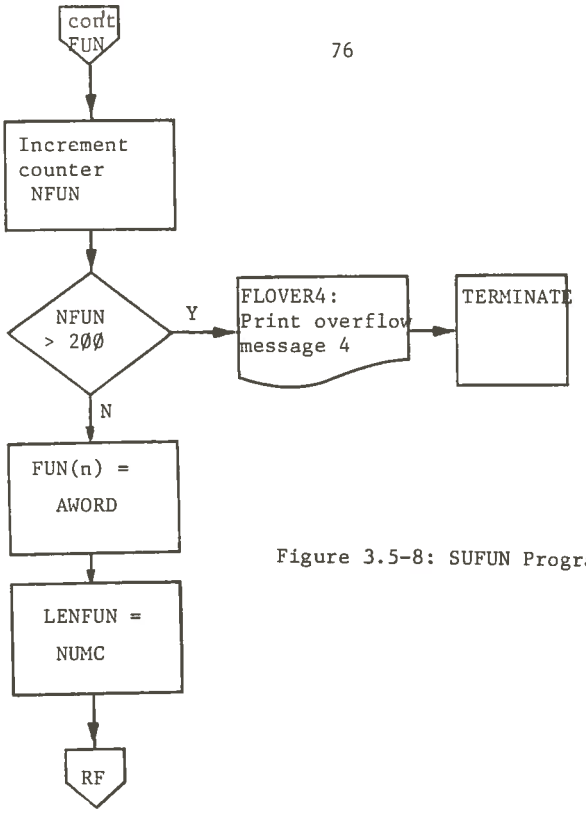
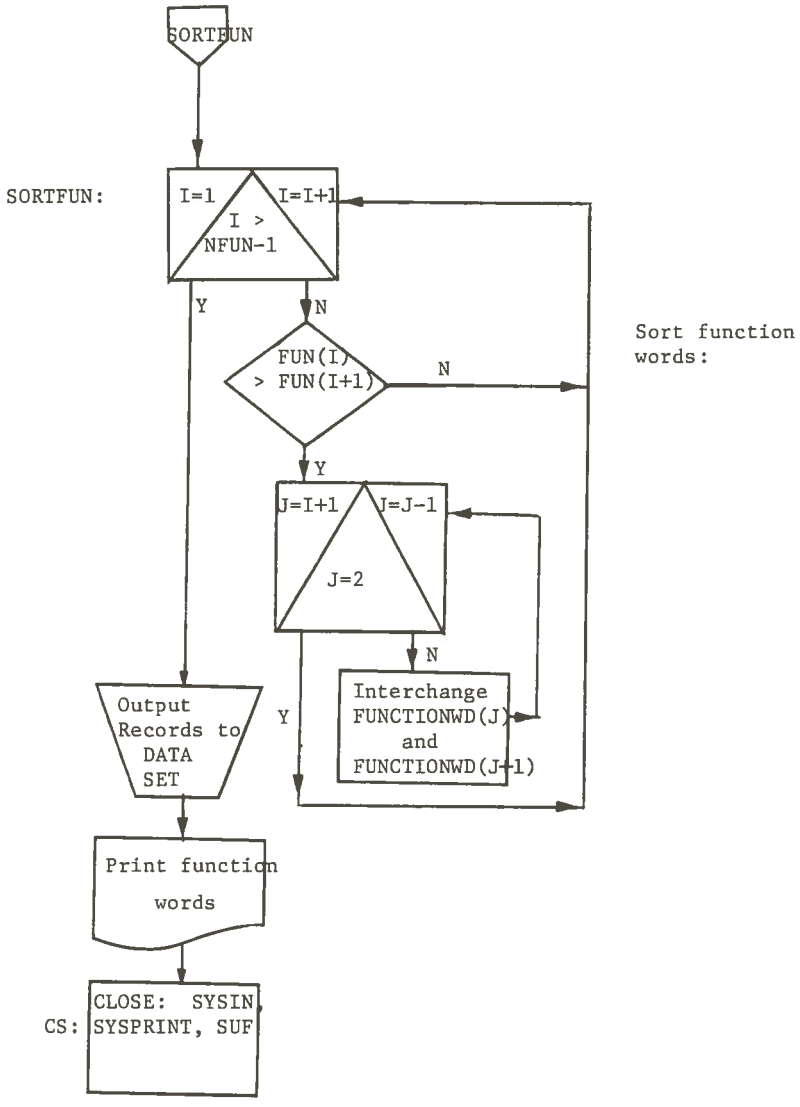


Figure 3.5-8: SUFUN Program: Main Routine



Sort function words:

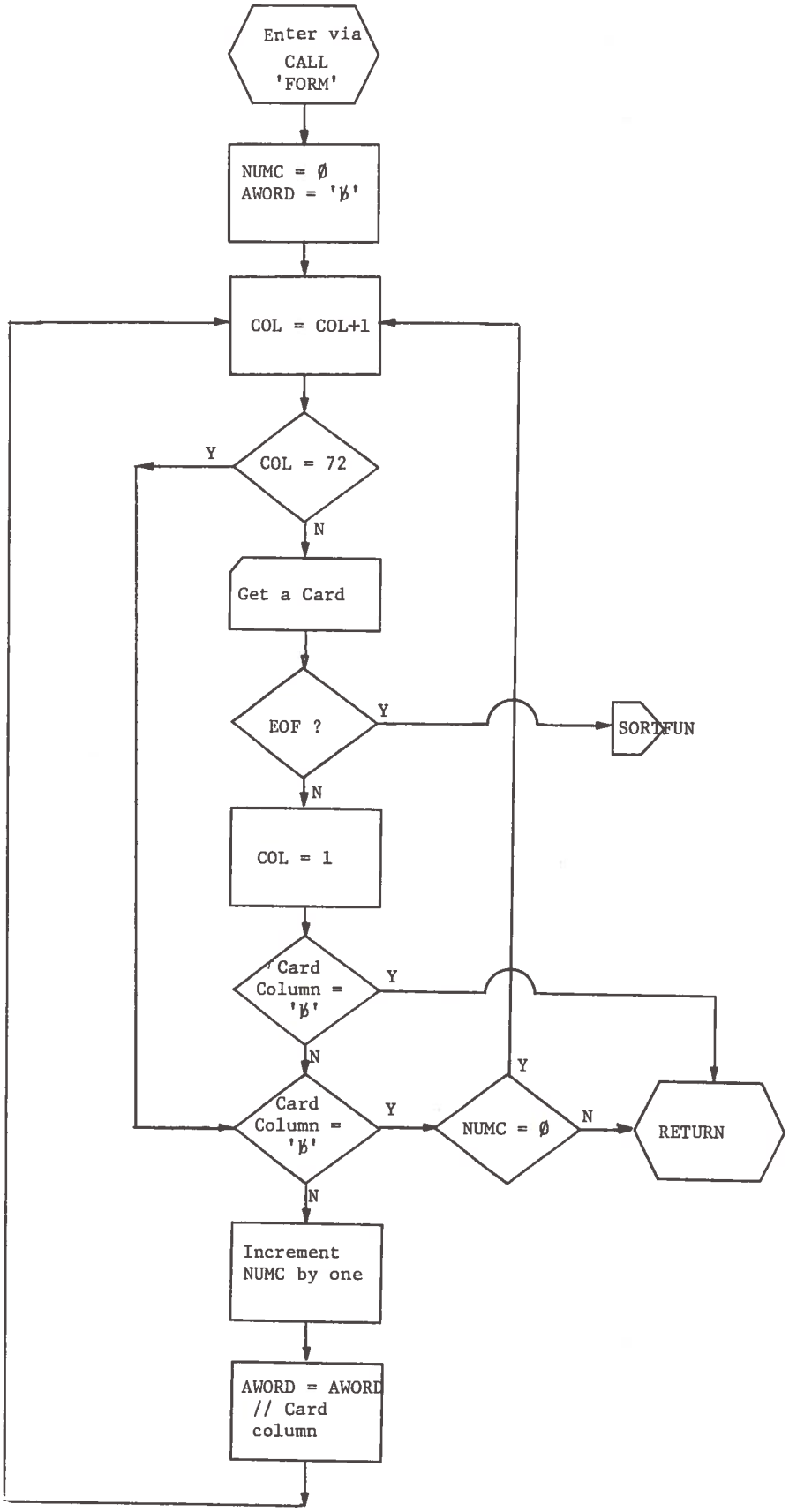


Figure 3.6: SUFUN Program: SUBROUTINE FORM: Procedure to distinguish words and/or punctuation using blanks as delimiters.

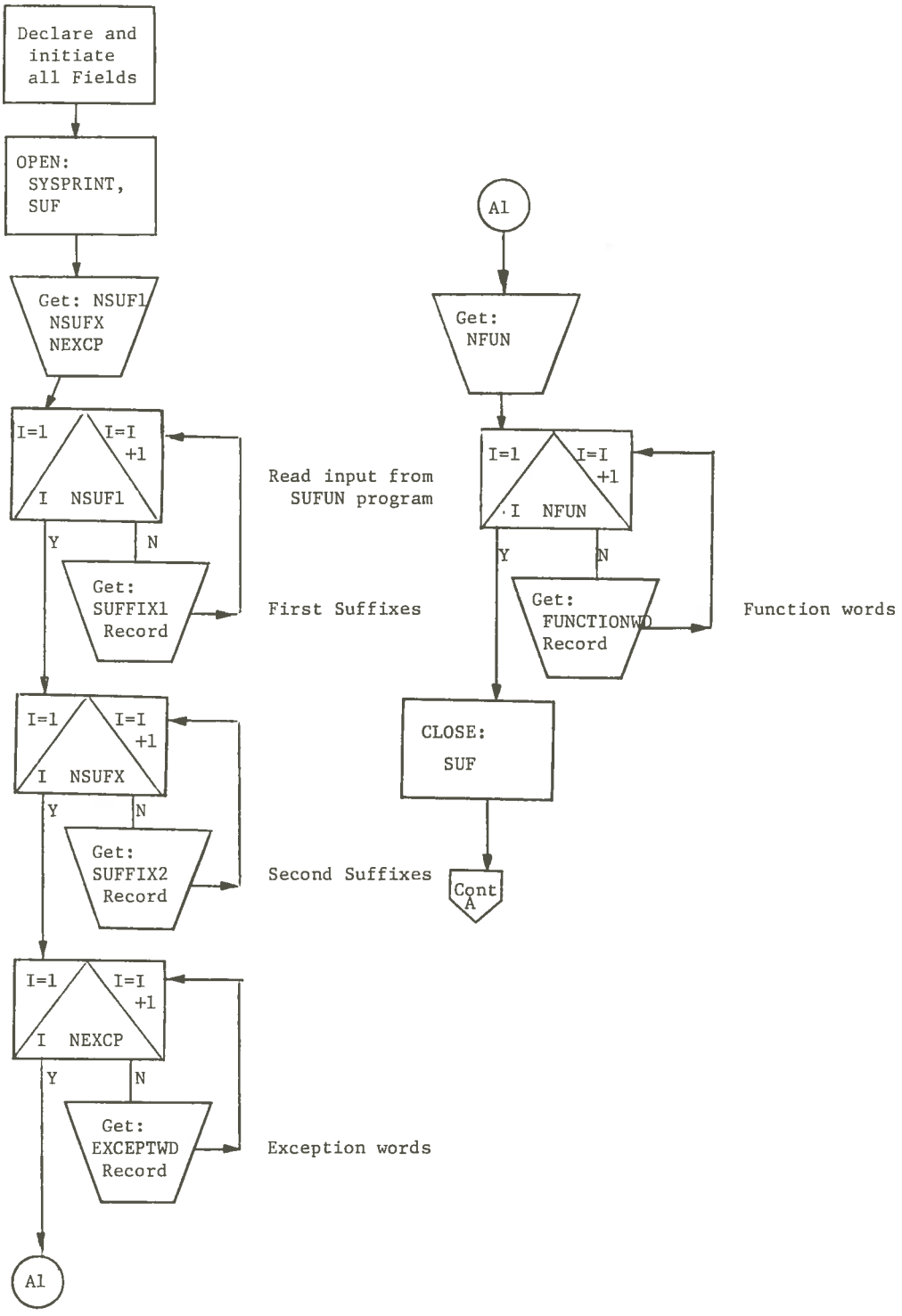


Figure 3.7: SUFFIX Program: Main Routine

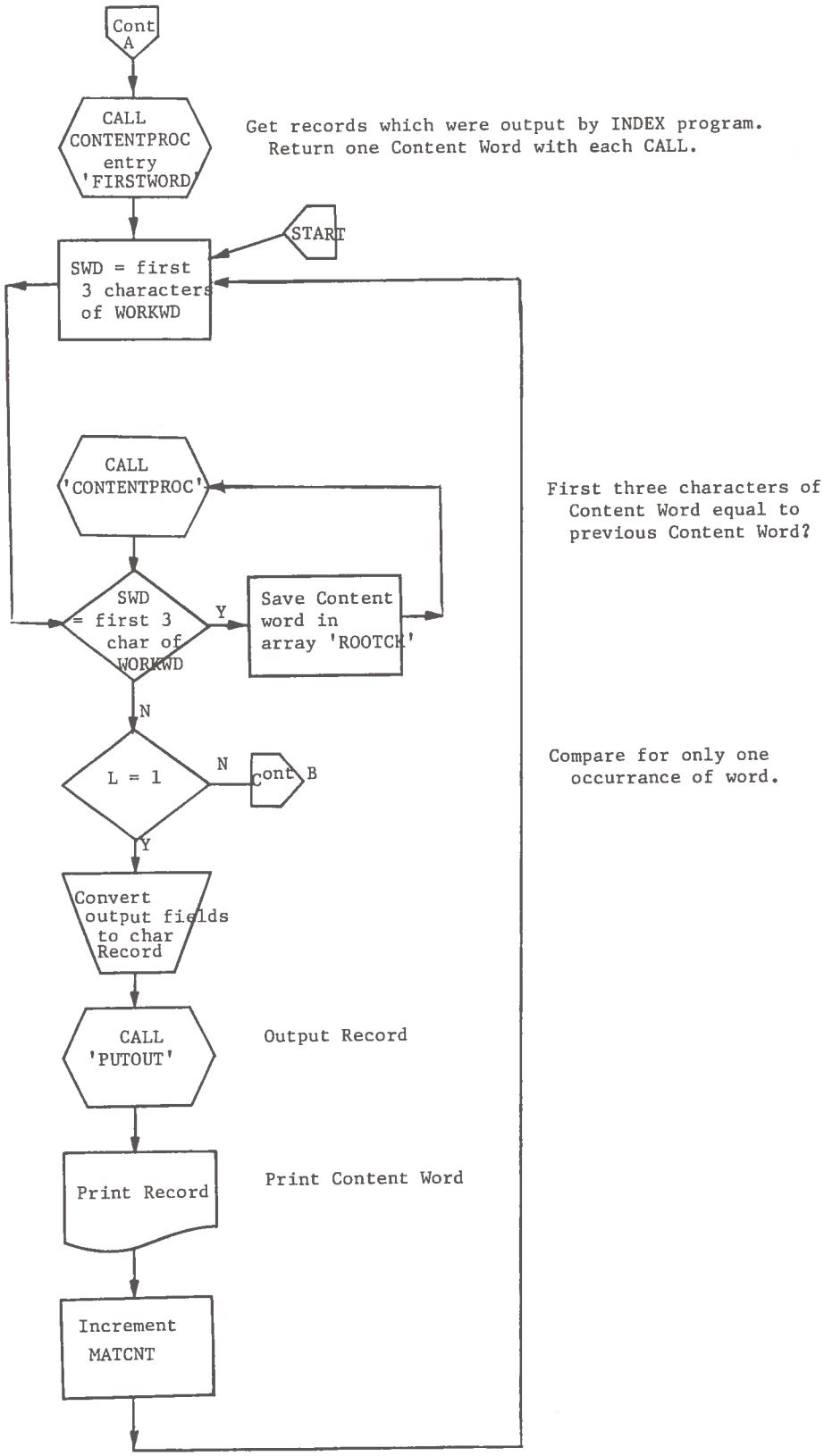


Figure 3.7-2: SUFFIX Program: Main Routine

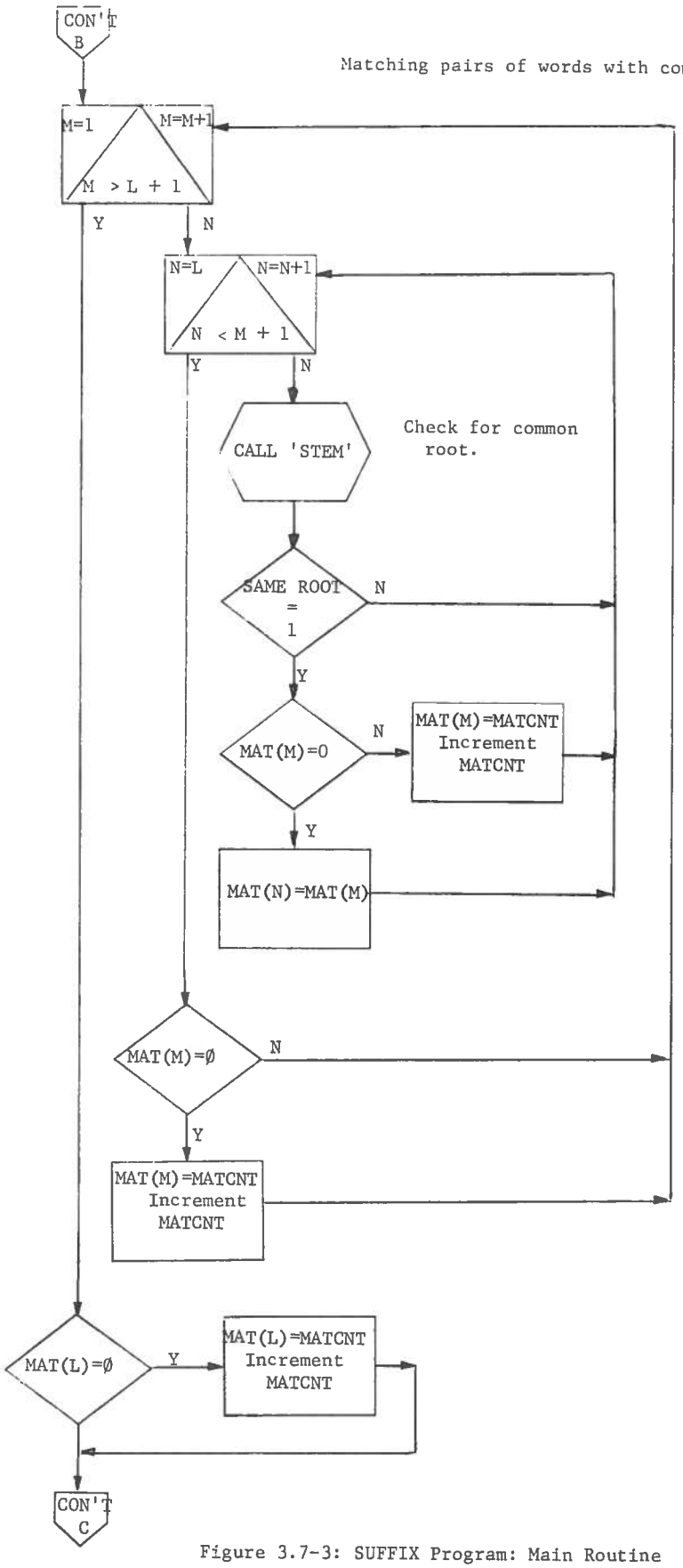


Figure 3.7-3: SUFFIX Program: Main Routine

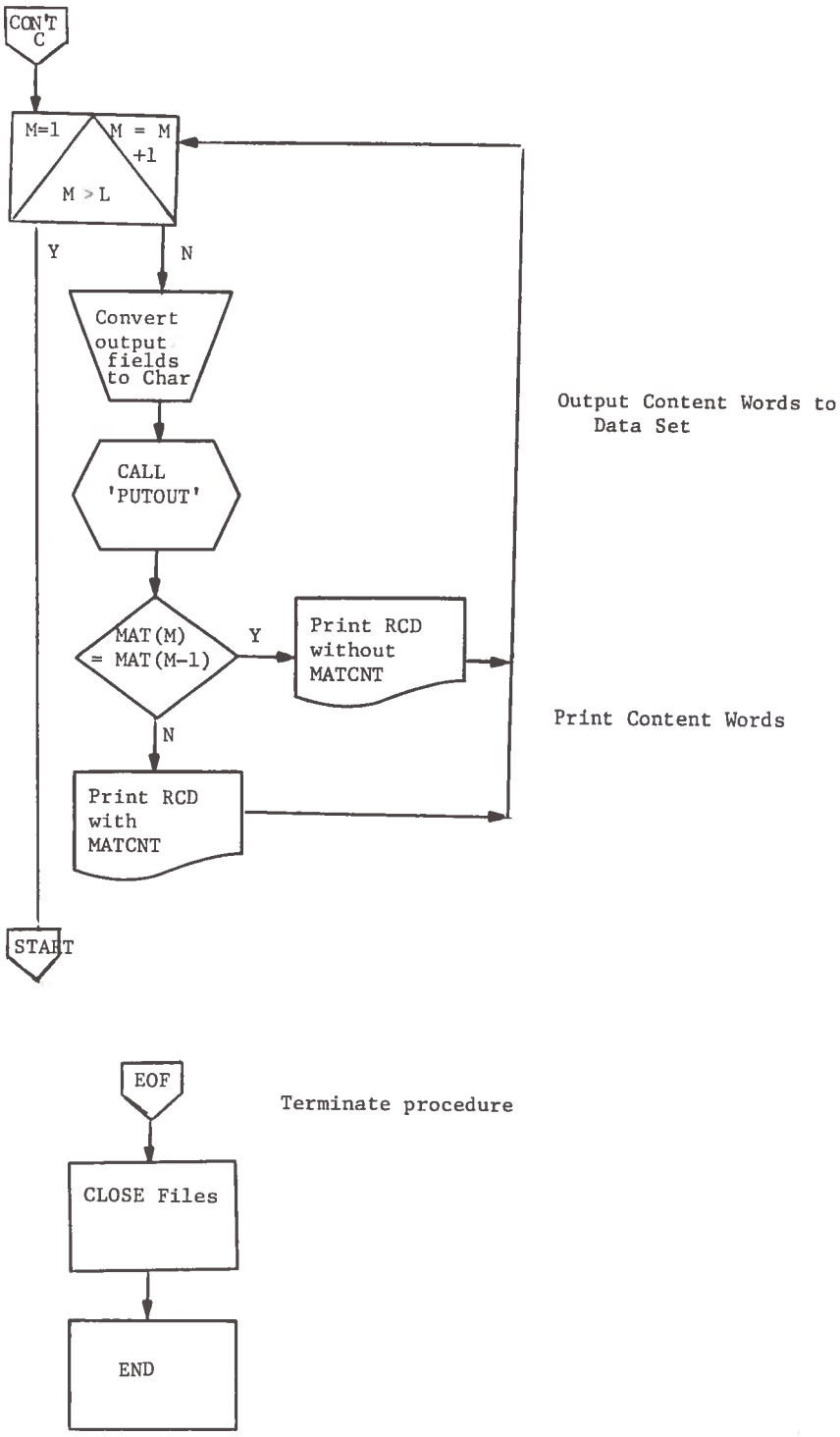


Figure 3.7-4: SUFFIX Program: Main Routine

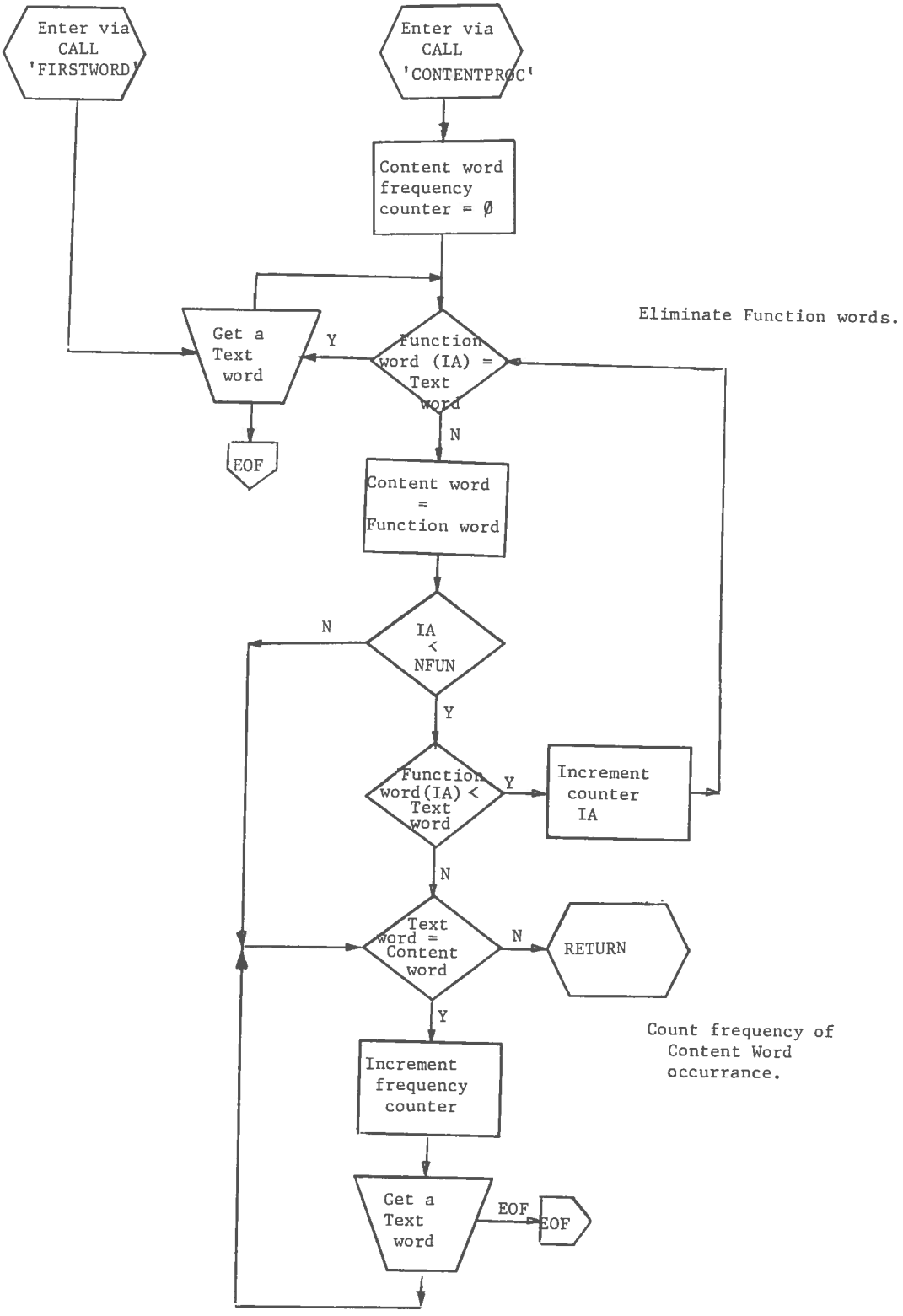


Figure 3.8: SUFFIX Program: Subroutine CONTENTPROC

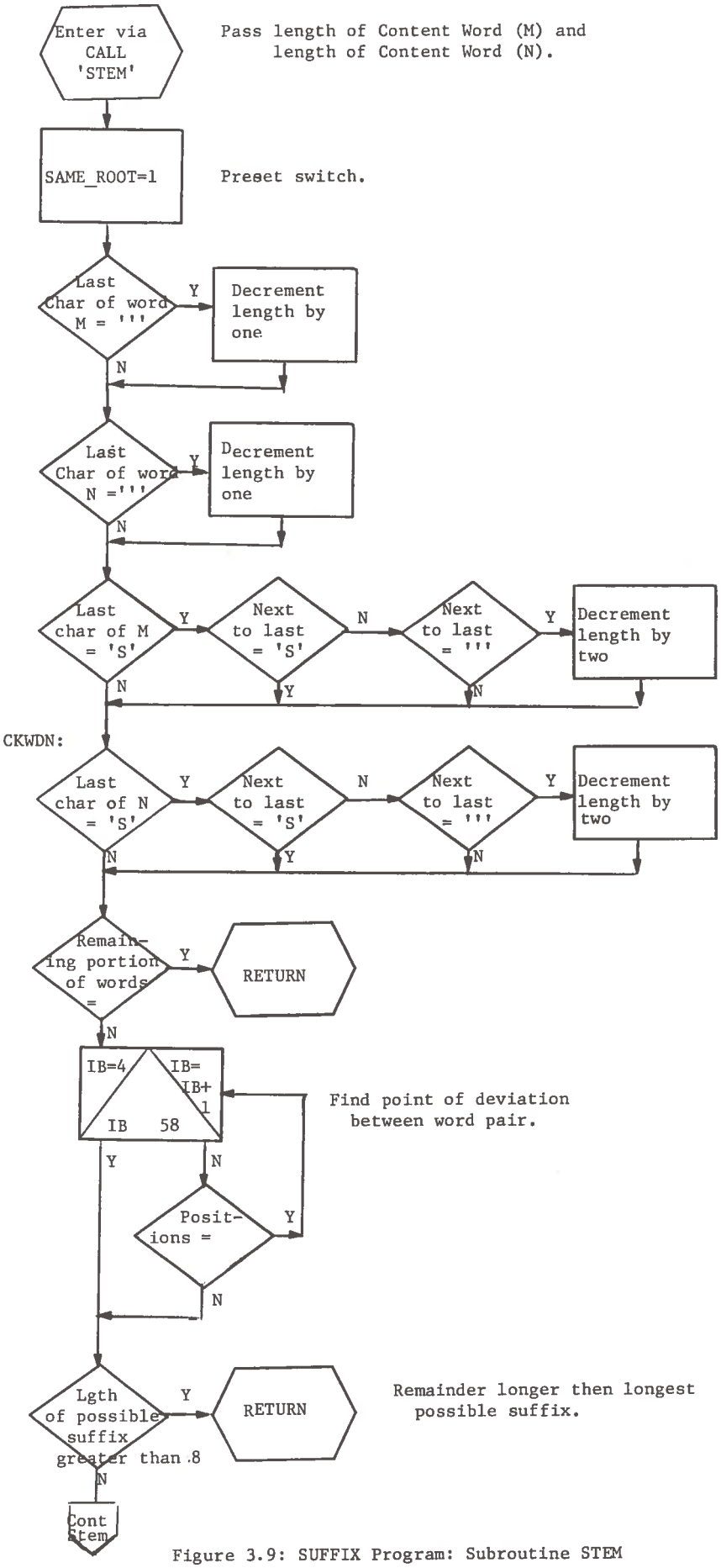


Figure 3.9: SUFFIX Program: Subroutine STEM

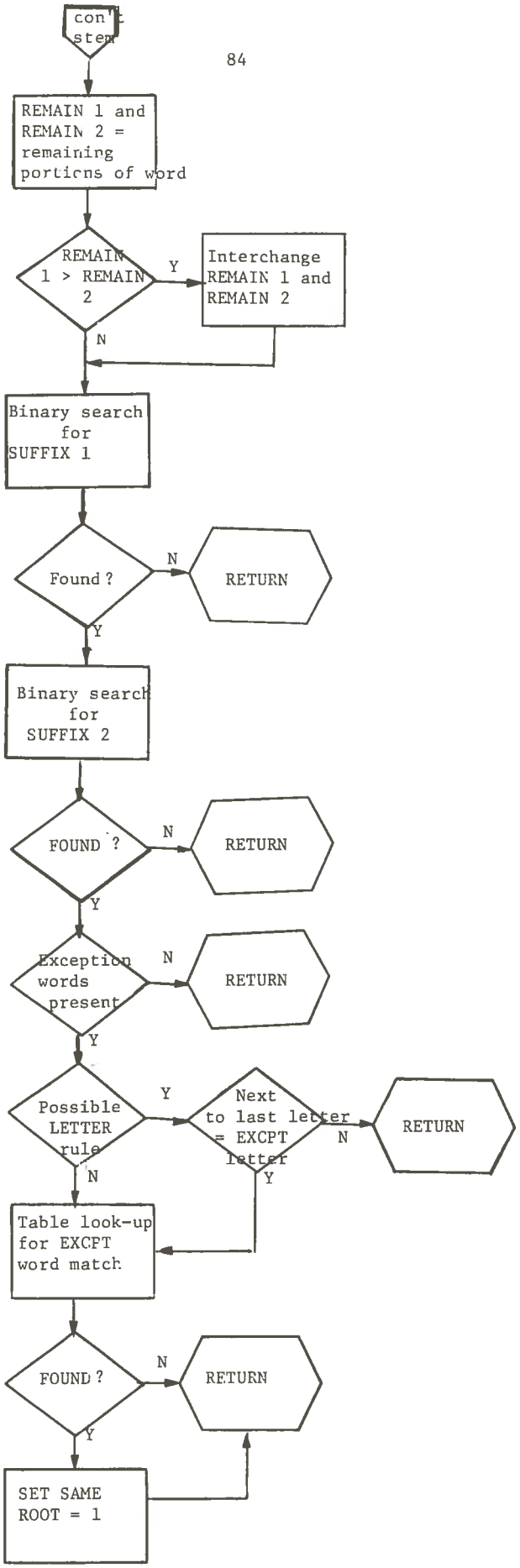


Figure 3.9-2: SUFFIX Program: Subroutine STEM

III. Program Documentation, Ring Structure VIA

by
William Buttelmann

A. Introduction

This portion of the report provides detailed description of the programs and data set organization which constitute VIA.

B. Data Sets

Text:

1. Input text. The input text is a simple character stream, in the format of standard printing conventions.

2. Formated text. The INDEX program produces a reformated text of variable length records. Each record has the following format.[12]

WORD LENGTH	MATCNT	FREQUENCY COUNT	WORD
4	4	4	1 - 58

Master Thesaurus:

The master thesaurus is actually composed of three data sets:

A vocabulary, which is an alphabetical list of each word type in the thesaurus;

A thesaurus, which is a list of pairs of pointers, one representing a word in the vocabulary and the other representing a semantic category; the thesaurus is sorted on category pointer;

A directory, which is a list of categories occurring in the thesaurus and of the initial position of each category. The directory is sorted on category number.

---

[12] In all record format diagrams, the numbers below each field specify the length of the field in storage positions (bytes) as represented in the IBM S/360.

3. Vocabulary. The vocabulary is a random-access data set which contains a record for each word in the thesaurus and for each word in the text. The latter records are inserted at the beginning of a processing run on a particular text by the PREP program which also deletes these record types left by previous runs on a different text. Thus, they are not permanent members of the master thesaurus. They will, however, become permanent members of the microthesaurus for a specific text. In addition, each record contains information necessary for thesaurus searching and enough text information to eliminate the need for further text searches. The text information is inserted by PREP. Each record has the following format:

DUM	KEY	MATCNT	SECTION	DIRECTORY POINTER	FREQUENCY COUNT	FLAG	WORD
1	4	4	4	4	4	1	1-58

DUM - Signals dummy records in the vocabulary.

KEY - Search Key for each record used by the random-access input-output system.

MATCNT - Developed and inserted by SUFFIX. It serves a dual purpose here:

1. To identify the root
2. To note that this word or a temporary entry with the same MATCNT is in the text.

SECTION - Number of current section of text. Inserted by PREP from an initial setup card called TEXTSECT card.

DIRECTORY POINTER - Index in the Directory of the first category entry containing this word.

FREQUENCY COUNT - Inserted by PREP, when ANALYSIS requests based on count are present.

FLAG - Used to indicate that there are other words in the VOCABULARY with this same MATCNT that are also in the text. The flag is 1 if this and others with the same MATCNT are in the text, 2 if others, but not this, are in the text. Otherwise the flag is blank. In case 2, the MATCNT is retained in the current record.

WORD - Up to 58 characters are allowed for the word.

4. Directory. The directory is a random-access data set which provides the linking between the vocabulary and the thesaurus. There is one record for each thesaurus category. Each record has the following format:

CATEGORY NUMBER	THESAURUS POINTER
8	4

CATEGORY NUMBER - The classification number of this category. Any combination of symbols, 8 characters or less is acceptable - such as the designations in Roget's Thesaurus (e.g. 101a.3).

THESAURUS POINTER - Position of the first record of this category in the Thesaurus. Records are ordered on THESAURUS POINTER. There is one dummy record at the end of the data set whose THESAURUS POINTER is set to one position beyond the end of the thesaurus data set and whose CATEGORY

NUMBER is 99999999. This record is needed so that the length of the last category in the thesaurus can be computed.

5. Thesaurus. The actual thesaurus is a random access data set. Each record represents an entry in the thesaurus - that is an entry of one word in one category. The format is:

DIRECTORY POINTER	VOCABULARY POINTER
4	4

DIRECTORY POINTER - Index in the Directory of the next category containing this word.

VOCABULARY POINTER - Index in the Vocabulary of the word for this entry. For a given word, the directory pointers thus provide the linking which "chains" through all the entries for that word. The pointer for the final entry of each word is set to point to the Directory entry for the first category containing that word. Thus, each chain is cyclic: a ring structure. The entire thesaurus, then, is a structure of interconnected rings.[13]

---

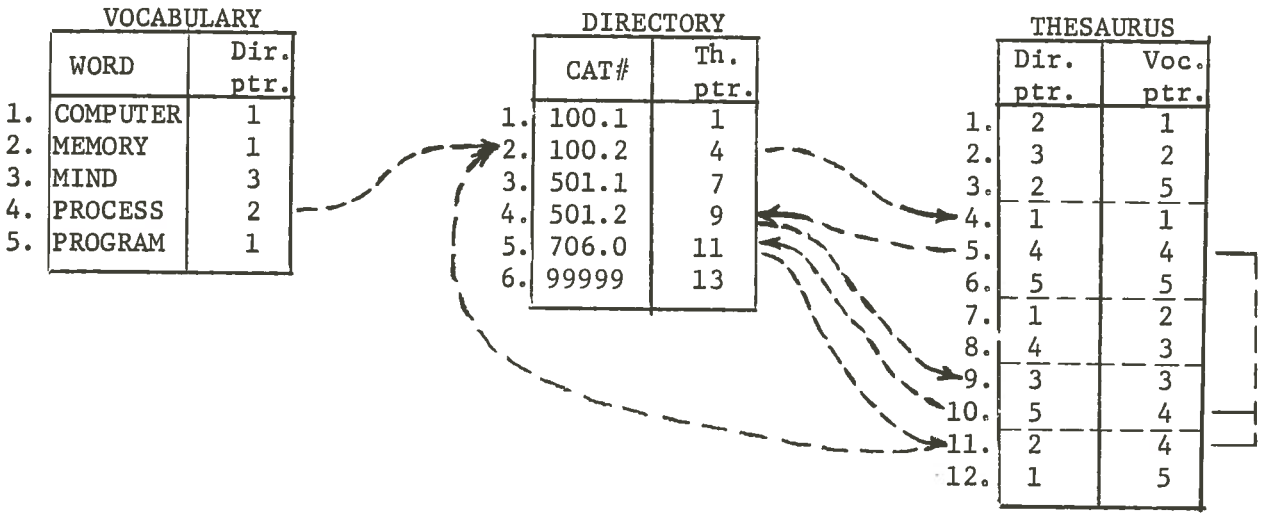
[13] Automatic Data Processing, Brooks, F. P., Jr., and Iverson, K. E., John Wiley and Sons, 1963, Section 6.2.

Searching is done by stepping around each ring, and for each entry in a ring, stepping around the ring linked with the first one by that entry. Because the rings are closed chains, each search eventually returns to the starting point in the ring; by saving the starting point, one knows when the search is complete.

The following simplified diagram may serve to illustrate the organization and linking of the Vocabulary, Directory, and Thesaurus. Suppose we have the following Thesaurus:

Category 100.1 Computer, memory, program
100.2 Computer, process, program
501.1 Mind, memory
501.2 Mind, process
706.0 Process, program

Then the data sets would be (we have reordered the Vocabulary records and elided unnecessary information for clarity):



As an example, consider a search for the word PROCESS. We first look it up in the Vocabulary. Its Directory pointer indicates that it first occurs in the second category in the Directory. We go to the second entry in the Directory and find there category 100.2 Its Thesaurus

pointer indicates that this category begins at position 4 in the Thesaurus. Since the next category begins at position 7, we know that category 100.2 is (7-4) = 3 entries long. Indeed, the entry for PROCESS is the entry "4-4" at position 5 in the Thesaurus, which is the second entry in category 100.2. To find the other entries for PROCESS, we proceed as follows: the Directory pointer at the first PROCESS entry is 4, indicating that the next entry for PROCESS is in the fourth category. We go to the Directory and find that this is category 501.2, which begins at record 9 in the Thesaurus and is 2 entries long. Indeed, there is an entry for PROCESS in record 10 in the Thesaurus. The Directory Pointer there points to the fifth category, which, according to the Directory is number 706.0, begins at entry 11 in the Thesaurus and is 2 entries long. Entry 11 is the one for PROCESS, and its Directory pointer indicates the second category, which was the first category in our ring. Thus, we have returned to our starting point, and the search around the ring is complete.

One may ask why the Directory pointers do not point directly to the next entry in the Thesaurus, rather than back to the Directory. The answer is that the category numbers are needed for printing and, more important, for each category, we suspend the step to the next link in the ring and initiate a ring search on every word in the category. Thus, we need to scan each category from top to bottom.

6. Keys. This file is a list of words, each of which initiates a search of the Thesaurus. In addition to the keying information, each key contains the entire Vocabulary entry for the key word. This information eliminates a later search of the Vocabulary. Each record has the

following format:

VOCAB entry less WORD	KFLAG	MODE	VOCAB POINTER	REQUEST NUMBER	WORD
22	1	1	4	4	1 - 58

KFLAG - An asterisk here denotes that this key has appeared in a previous section of text, but does not qualify as a key in the current section. Otherwise, this field is blank.

MODE - The search-and-print mode indicated on the ANALYSIS request card that generated this key is entered here. See the discussion of the THESAUR for the meaning of MODE.

VOCAB POINTER - Location of the Vocabulary record connected with this key.

REQUEST NUMBER - Identifying number taken from the ANALYSIS request card used to generate this key.

WORD - The key word.

C. Programs

This part of Section Two is separated into three parts, corresponding to the three parts in the structure of VIA described on pp 19 ff.: 1. Index; 2. Data Preparation; 3. Thesaurus Search and Construction.

1. Index. (See pages under Program Documentation)

2. Data Preparation.

This phase of the VIA system performs all the data preparations necessary to execute the searching and printing functions done by THESAUR in the third section. These include processing the TEXT and VOCABULARY files to complete certain information in them; reading the file of ANALYSIS requests submitted by the user, editing them for errors, interpreting them, and

setting the proper codes to cue the analysis and displays requested; and building or updating the file of KEYS which key the search-and-print action of THESAUR, and which form the nucleus of the microthesaurus for the given text.

The following steps give the logical sequence of events in this section. The reader will note that many of the functions listed in the paragraph above are performed somewhat in parallel, as the process proceeds. The result is that only one sequential pass of the text and one of the VOCABULARY are needed. Thereafter, the text is no longer needed and the VOCABULARY is accessed directly only when new words are required.

This phase has four distinct sections:

a) TEXTSORT, a utility sort of the TEXT file.

b) PREP, a PL/1 program.

c) KEYSORT, a utility sort of the KEYS file.

d) UPDATE, a PL/1 program.

a) TEXTSORT: This is a separate job step which sorts the TEXT file in alphabetical order on the word field.

b) PREP: This is a PL/1 program and is the major part of Section 2.

1. The first part of PREP, labelled REQUEST, is a self-contained section of coding that reads the control cards and request cards that establish parameters for the rest of the process. All cards are edited and any with incorrect or inconsistent entries are rejected, and diagnostic messages are printed. Correct cards are entered into a table, REQUEST, for subsequent processing. By omitting the GO AHEAD cards, described below, it is possible to rerun this section of PREP before proceeding with the rest of VIA. Thus, the deck of requests may be resubmitted with any rejected cards corrected. This allows one

to batch his analyses, which will save considerable computer time. The formats of the three input card types are described below. They have a "free format", in that each entry may begin in any column, and, except for the identifying entry, which must come first, the entries (i.e., parameters) may be typed in any order. Where blanks occur, any number of blanks is allowed.

The input cards are:

TEXTSECT card, which identifies the section of text currently being analyzed.

ANALYSIS card, which specifies the parameters for a complete text analysis and thesaurus search and print.

GO AHEAD card, which provides a positive means of causing text processing to begin after the ANALYSIS cards have been read and edited.

A typical deck would be composed of one TEXTSECT card, several ANALYSIS cards, and possibly a GO AHEAD card.

Since the parameters on these cards control all the functions of VIA, an understanding of their meaning is essential to successful use of the capabilities of this system. The following is a description of these parameters and their use in the system:

TEXTSECT card - This card is mandatory and must be the first card. The only entry required on it is

TEXTSECT = n;

n is the number of the current section of text being processed. It must be a positive integer. Optionally, one may enter any set of characters in the remainder of the card following the semicolon. These characters

will be printed as a heading at the top of the first page of output.

ANALYSIS request card - There may be up to twenty of these cards. Each specifies one complete text analysis and thesaurus printing. Its identifying entry is

ANALYSIS n;

This entry must be the first one in the card. n is an arbitrary number identifying the analysis and will be printed above each key word, thus identifying all the output for this analysis. The other entries are the following parameters; they may appear in any order, and each must be followed by a semi-colon, as shown.

TYPE is the only parameter required on all cards. The others vary. Different analysis types require different parameters. Familiarity with the various analysis types, described below, should make the choice of parameters clear.

The type parameter:

TYPE = 'n';

The type parameter actually provides a means for specifying how the key words for the analysis are to be chosen: whether by word or by category and whether by count or by fiat.

n specifies one of the following analysis types:

TYPE '1' - Frequency by category. 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 for the threshold, should be specified by a COUNT parameter in the same card. The program will total the number of textual occurrences of every word in each category. If a word occurs in more than one category, its total will be added to each. Every category whose total number of occurrences, which is the total of all occurrences of all words in the category, is equal to or greater than the threshold, will be used to key a search. This is accomplished by generating a KEY entry for each word in the category. 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' - Frequency by root. Every word 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 should be specified in the COUNT 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 2. The system chooses significant content in the text based on the high frequency of word roots.

Type '3' - Category. All words in a certain category are used as search keys. The category must be specified by a CAT parameter in the same card. A KEY entry is made for each word in the category. 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 in the WORD parameter. It will be used to key a search. This type of analysis is the fastest available, 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 the parameter word.

The mode parameter:

MODE 'a' - a is the letter A, B, C, D, or E, and denotes one of the five search modes described in the description of THESAUR in Section C.

COUNT = 'n' - n is the count threshold. It may be any real number.

The word parameter:

WORD = 'word' - word is any word. If it is neither in the master thesaurus nor in the text, no output will appear.

The category parameter:

CAT = 'category' - category is the identifying label for any category that appears in the thesaurus; more specifically, it is any category that is listed in the DIRECTORY data set.

GO AHEAD card - This card has the words "GO AHEAD" typed anywhere in the card. Nothing else may be entered in it. A GO AHEAD card is mandatory, following the last ANALYSIS card, if it is desired to continue with the rest of the VIA processing after the ANALYSIS requests have been processed. If no GO AHEAD card is present, the program will cancel the rest of the job after reading and diagnosing the ANALYSIS requests.

2. Sort the REQUEST table on TYPE. This is an internal shuttle sort done by the coding labelled SORT_RQS.

3. The next small section of coding, labelled TYPESEARCH searches the REQUEST table for type and analyses, and identifies their positions in the table by saving their initial and final indices.

4. The next part of PREP, labelled UPDATE_VOCAB, processes the TEXT and VOCABULARY files together, sequentially. The first part of this coding is the program SUFFIX, which is described in the earlier documentation

of VIA. It has been modified here to read the Vocabulary data set in parallel with the TEXT. SUFFIX identifies words with the same root and enters a root-identifying number, MATCNT, in the field TMATCNT, if the word is in a text record, and in VMATCNT, if the word is in a Vocabulary record.

At the same time, temporary entries are made in the Vocabulary for each word in the TEXT that is not already in the Vocabulary. These are marked by setting their VFLAG to 3. If such a temporary entry is made, the Vocabulary is then searched for other words with the same MATCNT as the temporary word. The VFLAG of each of these is set to 1 if the word for that entry also appears in the TEXT and to 2 if the word does not. If there are no words in the Vocabulary with the same MATCNT, the warning message THE TEXT WORD "word" HAS NO RELATIONS IN THESAURUS will be printed. This message informs the research user that the text word will not appear in any printed output nor in any micro-thesaurus. He may wish to examine the message list and make suitable additions to the thesaurus.

Another process executed while the text and vocabulary are read is updating the text section entry (VSECT) in the Vocabulary. If the TEXT-SECT number from the TEXTSECT card is 1, then this is the first section of a new text being analyzed. In this case, all previous VSECT entries are blanked out and all temporary vocabulary entries, marked by VFLAG = 3, are erased when they are read. As each record is returned to the Vocabulary, and as new temporary entries are made, if the word appears in the current section of text, TEXTSECT is entered in VSECT. Thus, as progressive sections of text are read, Vocabulary contains the section of text in which each word first appeared, and information to show whether it appears in the current section.

The counting process is also done as the text is passed, and the count for each word is entered in VCOUNT in the Vocabulary. Each time the total of all words in a MATCNT group is computed, a scan of the type 2 requests is made. For each type 2 request whose threshold is met by the MATCNT total, a KEY is generated for every word having that MATCNT.

5. The next section of coding, labelled BLDKEYS, reads the remainder of the table of REQUESTS and builds the rest of the KEYS. If there are any type 1 requests, the coding labelled TYPE1 executes the following process: the Directory is read sequentially, and each category in the Thesaurus is examined. All words in the category are located and their counts, from YCOUNT (the count field in the Vocabulary record) are summed. For each category whose sum of counts is thus obtained, the type 1 requests are scanned. For each type 1 request whose threshold is met by the total of the VCOUNTs, a KEY is generated for every word in the category.

Type 2 requests have already been processed.

Each type 3 request causes the category specified in its CAT parameter to be located in the Directory. Since the current program is written for a general data set organization, a binary search is used to find the category position in the Directory. This may actually prove quite slow for some large size directories, and it may be advisable to replace the search with a key transformation. On the other hand, if the number of requests is kept small, any time saved would be negligible. Once the category is located, all words in the category are sought through the Thesaurus and a KEY entry is made for each word.

Type 4 requests cause a KEY to be entered for the word specified in the word parameter.

After the REQUEST table has been processed, all the key words for thesaurus searches have been found. These comprise the KEYS table, and it

is now completely built.

6. The last section of PREP begins with the label KEYS_COMPLETE. All that remains to do in PREP is to sort the KEYS table, first on ANALYSIS (REQUEST) number, then on WORD. If the table is small enough to be held in memory, it is sorted by the last section of coding in PREP, and a user completion code is passed to the Operating System causing the next job step to be skipped. Finally, the sorted KEYS are written onto a temporary data set. If the KEYS table is too large for memory, a portion of it has already been written on the temporary data set. In this case, no sorting is done by PREP; the last of the KEYS are simply written onto the data set.

c) KEYSORT: This is the third phase of the data preparation section of VIA. It is an optional job step and will be executed by the computer's Operating System only if the last part of PREP, the KEYS internal sort, was not executed. This phase is a utility sort of the temporary data set containing the KEYS.

d) UPDATE: This is a PL/1 program which produces the updated file of KEYS used by THESAUR to cue thesaurus searches. When UPDATE begins processing, there are two data sets of KEYS: 1) the temporary data set just produced by PREP, and 2) the data set of KEYS from analyses of previous sections of text. This program produces an updated set of KEYS which is the logical union of the two input data sets in the following manner: The two data sets are compared sequentially. First, all the KFLAGS in the old set are blanked out. If a KEY in the new set is not in the old set, it is added to the old set, and a period (.) is put in its KFLAG. Thus, all words appearing as key words for the first time will be preceded on the printout by a period (e.g. . KEY). If a KEY in the old

set is not in the new, an asterisk (*) is put in its KFLAG. Thus all words appearing as key words in previous sections, but not in this section, will be preceded by an asterisk on the printed output (e.g. *KEY). Thus augmented, the old KEYS data set constitutes the updated KEYS. This is the data set passed to THESAUR. The data set of KEYS produced by PREP is the nucleus of a microthesaurus for the current section of text. The updated KEYS produced while processing the final section of text is the nucleus of the microthesaurus for the entire text.

3. Thesaurus Search and Construction.

This section consists entirely of the PL/1 program THESAUR. The program searches the thesaurus for word-relations and prints out the relationship pattern in a tree format. It operates sequentially on the KEYS file. Each key causes one complete search-and-print operation, and the KEY word serves as the root of the relationship tree.

The tree to be printed may be pruned, depending on the MODE parameter of the ANALYSIS request card. That mode is passed from PREP to THESAUR in KMODE in the KEY. Two other areas - MATCNT and VFLAG - determine exactly which words in the Vocabulary are in the text. (All words in the text are in the Vocabulary.) They are used as follows: If MATCNT = 0 neither this word nor any root-related words are in the text. If MATCNT $\neq$ 0 one of two situations can occur: If VFLAG = 1 then this word and others in the Vocabulary with the same MATCNT are in the text; If VFLAG = 2 then others in the Vocabulary with the same MATCNT are in the text, but this word is not. In either case, the other words were temporary additions made by PREP, and their VFLAG is set to 3.

One other piece of data may determine printing: If the KFLAG in a KEY is *, this signals that the word has been a KEY in some previous section of the text, but is not a KEY in the current section. It will therefore be used

to key a search and will be printed, preceded by an asterisk, regardless of the search mode. A final printing rule holds for all nodes: No node (word or category) will appear more than once in a path of a tree. This eliminates an obvious redundancy.

Three data areas of interest are the vectors PATH, WORDSP, and CATSP.

PATH is a vector of indices representing the current path the search-and-print is working down. Because of the interlocking ring structure of the Thesaurus (word ring-category ring-word ring-category ring, etc.) the 1st 3rd, 5th, 7th, and 9th nodes will be word pointers and the 2nd, 4th, 6th, 8th and 10th will be category pointers. The chief purpose of PATH is to provide a means for enforcing the overall printing rule of eliding any node (and the subtree rooted at it) that has already appeared in the path.

WORDSP and CATSP are vectors of character strings. The strings are the actual words and category numbers to be printed. These vectors are needed, because, for some search-and-print nodes, the decision to print cannot be made when the node is found. When a decision to print is made, these vectors provide the necessary print information, and extra readings of the Vocabulary and Directory are eliminated. These vectors are organized so as to make their entries correspond with the entries in PATH. Thus, WORDSP(I) = VWORD(PATH(I)) and CATSP(I) = DCAT(PATH(I)). As a consequence, half of each of the -SP vectors is unused. This is a small expense of memory to save the time that would otherwise be needed to compute their indices; i.e., only one index serves for all three vectors.

Other data items in the index are: CURCAT, the index in Directory of the first category. This location marks the starting point of the ring search. PRINTDX, the last position in PATH (and hence in WORDSP and CATSP) that has been printed. LEVEL, the current level of recursion in the program. The

algorithm is a main procedure with one recursive subprocedure, which functions as follows:

Main Procedure:

1. Get the next KEY. Stop when the KEYS are exhausted.
2. Determine from KMATCNT, KFLAG, and KMODE if it should root a search. If not, go to step 1.
3. Print a heading identifying the new search.
4. Determine from KMATCNT, KFLAG, and KMODE if this KEY should be printed now or saved in WORDSP for possible later printing.
5. a. If the KEY is to be saved, put it in WORDSP, preceded by KFLAG, and set PRINTNDX to 0.
   b. If the KEY is to be printed, print it preceded by its KFLAG, and set PRINTNDX to 2.
6. Initialize PATH to zero and set PATH(1) = KWORD.
7. Set LEVEL=1, indicating the highest level of recursion.
8. Initialize parameters for the recursive subprocedure as follows:

   PARM1 = KWORD
   PARM2 = PARM3 = 1st category in the ring of categories containing this word.

9. Call the recursive subprocedure.
10. Upon return from the subprocedure,go to step 1.

Subprocedure (recursive):

This procedure steps through the ring of categories containing the word passed to it, beginning at the category passed to it. For each category in the ring, it steps through the ring of words in that category. For each word in that ring it executes a call to itself, thus creating a new search at a new sublevel.

1. See if a complete circuit of the ring of categories has been made. If

so, go to step 14. If not, then we are at the next category in the ring.

2. See if this category is already in the PATH. If so, bypass it by

   a. finding the next category in the ring,

   b. going to step 1.

3. At this point, we know we have a new category. Put it in the next position in the PATH.

4. Determine whether to print the category now or save it. The decision is based on the disposition of the word passed to this subroutine from the next higher level; this category is to be printed if and only if that word was printed.

5. a. If the category is to be saved, put it in the next available position in CATSP.

   b. If the category is to be printed, print it.

(At this point, we begin to step around the ring of words in this category. Steps 6 - 12 form a loop which does the iteration.)

6. Find the first/next word in the ring of words. If the ring has been completely searched, go to step 13.

7. Determine whether or not this word is to be bypassed in the search. It is if this is so indicated by KMODE and VMATCNT, or if the word is already in the PATH. If it is to be bypassed, do so by going back to step 6.

8. At this point, we know we have a new word. Put it in the next position in the PATH.

9. Determine from VMATCNT and KMODE whether this word is to be printed now or saved.

10. a. If the word is to be saved, put it in the next available position in WORDSP.

b. If the word is to be printed, print all the entries that have been saved in WORDSP and CATSP, then print this word, and set PRINTNDX to key printing all categories in the next level of recursion.

11. If we have more levels of recursion to go through, we do so here.

a. Increase LEVEL by 1.

b. Set parameters for the subprocedure call as follows:

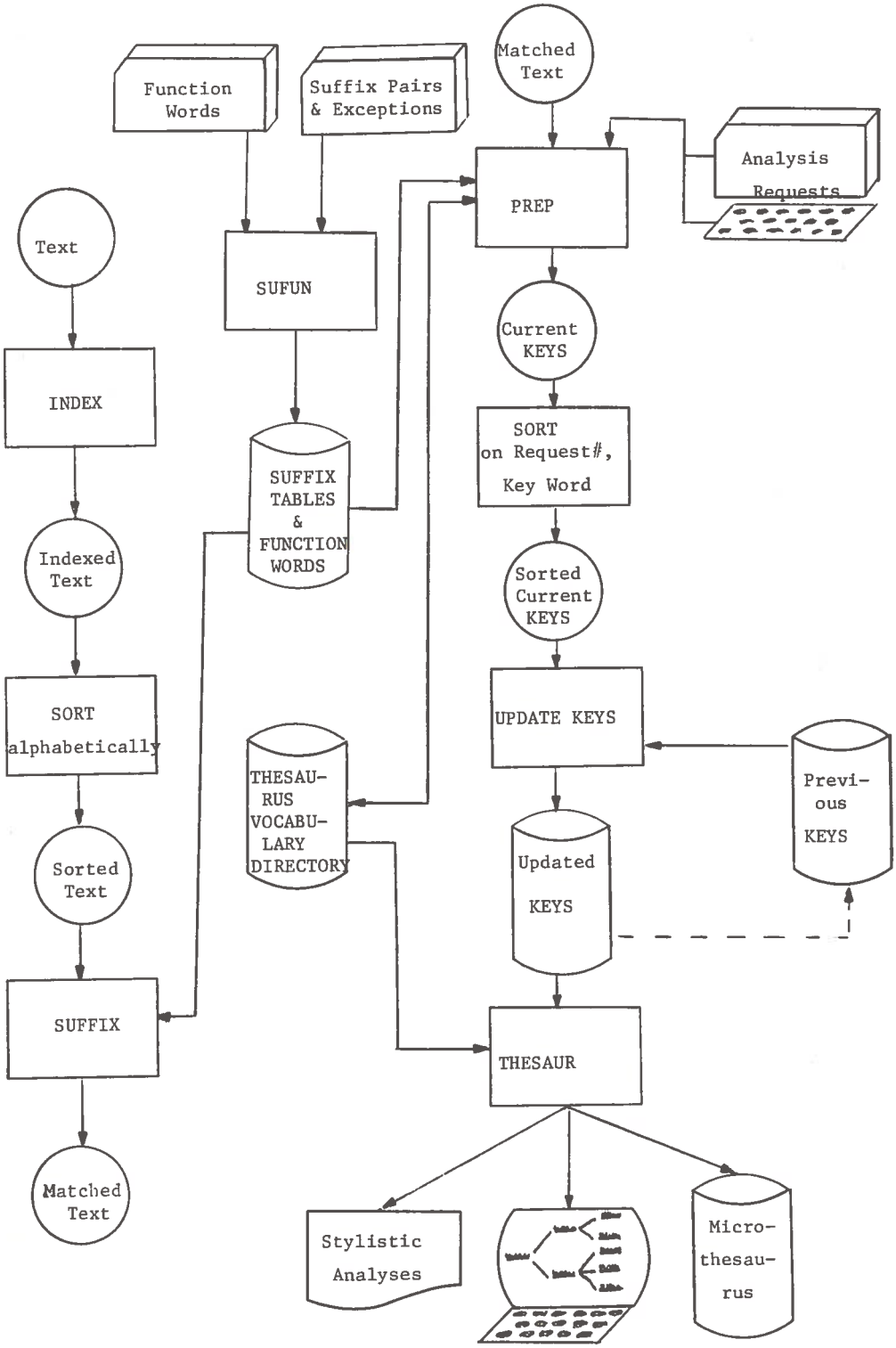
PARM1 = current word
PARM2 = current category (marks the head of the ring of categories to be searched on the next level.)
PARM3 = next category in the ring of categories to be searched.

c. Call this subprocedure.

12. Upon return from the subprocedure, or after bypassing step 11, go to step 6, to continue the ring of words. This step closes the loop of steps that search around a ring of words.

13. Here we have completed the circuit in a ring of words. Find the next category in the ring of categories and go back to step 1. This step closes the loop of steps that search around a ring of categories.

14. Here we terminate the subprocedure. Reduce LEVEL by 1. Remove the last entry from the PATH. Return to the step following the calling step.



Ring-Structured VIA

IV. Professional Activities of Project Personnel

Sally Y. Sedelow

Publications:

* "Stylistic Analysis," Automated Language Processing: The State of the Art, edited by Harold Borko, John Wiley and Sons, 1967. Co-author with Walter A. Sedelow, Jr.
* Stylistic Analysis, Third Annual Report, SDC Document TM 1908/300/00, March 1, 1967.

Papers/Seminars/Addresses/ etc:

* Speaker,"Opportunities for Computer-Aided Research at UNC," Institute for Research in the Social Sciences, Chapel Hill, February 1967.
* Speaker, "Computer-Aided Analysis of Language," Naval Reserve Unit, Chapel Hill, February 1967.
* Discussant, session of "Measurement in the Humanities: Can the Computer Help?" American Education Research Association, New York, February 1967.
* Chairman and Panelist, conference on "The Computer and Research in the Humanities," University of North Carolina, March 1967.
* Paper, "Computer-Aided Stylistic Analysis, University of Colorado, Boulder, July 1967.
* Paper, "Four Faces of Language Analysis," American Documentation Institute, New York, October 1967.

* Paper, "Categories and Procedures for Content Analysis in the Humanities," National Conference on Content Analysis, University of Pennsylvania, November 1967. With Walter A. Sedelow, Jr.
* Instructor, Study Institute on Computer-Aided Language Analysis in Education, American Education Research Association, Chicago, February 1968.

Activities:

* Consultant, Thomas Y. Crowell Company, 1966 --
* Co-editor, Journal for Computer Studies in the Humanities and Verbal Behavior, Mouton, The Hague, The Netherlands, 1966 --.
* Field Reader, U.S. Department of Health, Education and Welfare 1966 --.
* Member, Review panel, session on "The Humanities and Social Sciences," Fall Joint Computer Conference, 1967.
* Chairman, Special Interest Committee on Language Analysis and Studies in the Humanities, Association for Computing Machinery, 1968--.

Walter A. Sedelow, Jr.

Publications:

* "Stylistic Analysis," Automated Language Processing: The State of the Art, edited by Harold Borko, John Wiely and Sons, 1967. Co-author with Sally Y. Sedelow.
* "Computational Sociolinguistics," Research Previews, 14(2), 1967.

Papers/Seminars/Addresses/ etc.:

* Speaker, Institute for Research in the Social Sciences, Chapel Hill, February 1967.

* Speaker, session on "Measurement in the Humanities: Can the Computer Help?" American Education Research Association, New York, February 1967.
* Panelist, "The Computer in Aid of the Scholar," Conference on "The Computer and Research in the Humanities, University of North Carolina, March 1967.
* "Society and the Librarian: A Prognosis," North Carolina Library Association Biennial Conference, General Session Address, Charlotte, North Carolina, October 1967.
* "Categories and Procedures for Content Analysis in the Humanities," National Conference on Content Analysis, University of Pennsylvania, November 1967. With Sally Y. Sedelow.
* "Knowledge Systems," presentation to the University Committee on Communications, University of Pennsylvania, January 1968.
* "The Interdisciplinary Structure of the Research," Study Institute on Computer-aided Language Analysis in Education, American Educational Research Association, Presession #7, February 1968.
* "The Implications of Computers for Knowledge," The 'Cutting Edge' Symposia, (Danforth Foundation), University of North Carolina, Chapel Hill, February 1968.

Activities:

* Consultant, Computer-Aided Linguistic Analysis Project (sponsored by the U.S. Office of Naval Research), March 1967 --
* Consultant, Jacksonville (Illinois) State Hospital, February 1968 --
* Consultant, University Committee on Communications, University of Pennsylvania, January 1968 --
* Consultant, Manuscript Evaluation, John Wiley & Sons, Inc., 1967 --

* Consultant, Project Evaluation, Thomas Y. Crowell, Inc., 1966-67.
* Editorial Board, Social Forces, 1966 --.
* Editorial Board, Computer Studies in the Humanities and Verbal Behavior, 1966 --.
* Trustee, International Social Science Institute, 1966 --.
* Administrative Board, Frank Porter Graham Child Development Research Center, Chapel Hill, North Carolina, February 1968 --.
* Member, American Council of Learned Societies' Computer Applications Committee, February 1968.

Walter L. Smith

Papers published:

* "On the weak law of large numbers and the generalized elementary renewal theorem," Pacific Journal of Mathematics, 22, pp. 171-188, 1967.

Papers written:

* "Remarks on renewal theory when the quality of renewals varies," Institute of Statistics Mimeo Series #548, Sept. 1967, Chapel Hill.
* "Necessary conditions for almost sure extinction of a branching process with random environment," Institute of Statistics Mimeo Series #549, September 1967, Chapel Hill.
* "Some results using general moment functions," Institute of Statistics Mimeo Series #554, November 1967, Chapel Hill.

Papers presented:

* Paper #3 presented at the 36th Session of the International Institute of Statistics, August - September, 1967, Sydney, Australia.

Joan Nancy Bardez

* Master's Thesis, "Linkage-Loader Design In Computer Operating Systems," University of North Carolina, Chapel Hill, 1968.

William G. Hickok

* "A Technique for Structuring a Formated File System Retrieval Request from Areas Delineated on a WAC or ONC Utilizing the Gerber Inverse Plotter," NAIRY 662, NARTU Andrews, for LANT INTELCEN, October 1967.

APPENDIX

```
1                INDEX:  PROC (PARM) OPTIONS(MAIN);                                  113

              /*  THE INDEX PROGRAM INDEXES TEXTUAL WORDS AND/OR PUNCTUATION MARKS */
              /*  PUNCHED ON DATA PROCESSING CARDS.  WORDS AND/OR PUNCTUATION MUST */
              /*  BE DELIMITED BY BLANKS.    THE PROGRAM INDEXES THE TEXT WORDS,   */
              /*  ETC. BY VOLUME NUMBER, CHAPTER NUMBER, PARAGRAPH NUMBER,         */
              /*  SENTENCE NUMBER, AND WORD NUMBER WITHIN SENTENCE.                */

2                  DECLARE PARM CHAR (40) VARYING;  /* DATA FROM JCL 'EXEC CARD */

3                  DECLARE CARDIMAGE (80) CHAR (1) STATIC,
                           TEMP    CHAR (10) INITIAL (' '),
                           UHLAB CHAR (4) INITIAL (' '),/* TYPE OF PROCESSING   */
                           PRNTCTRL CHAR (5) INITIAL (' '),  /* PRINT, NOPRT PARM */
                           STAGEDIRECT  CHAR (15) INITIAL (' '),
                           TOVO FIXED (4) INITIAL (1),  /* TOTAL VOLUMES        */
                           TOCH FIXED (4) INITIAL (1),  /* TOTAL CHAPTERS       */
                           TOPA FIXED (6) INITIAL (1),  /* TOTAL PARAGRAPHS     */
                           TOSE FIXED (6) INITIAL (0),  /* TOTAL SENTENCES      */
                           TOWO FIXED (6) INITIAL (0),  /* TOTAL WORDS          */
                           L LABEL (L7,L8,L9,L10);

4                  DECLARE   /* USED IN SUBROUTINE 'FORM'.                        */
                           SEQUENCE CHAR (8) INITIAL (' '),
                           LINE     FIXED (4) INITIAL (0),
                           LINEA    FIXED (2) INITIAL (0),
                           COL      FIXED (2) INITIAL (73),
                           COLIM FIXED (2) INITIAL (72),  /* NUMBER OF COLUMNS  */
                           WRDCONT FIXED DEC (1) INITIAL (0),
                           LINECONT FIXED DEC (1) INITIAL (0);
                                                 /* TO BE PROCESSED     */
5                  DECLARE     /* USED IN OUTPUT RECORD.                          */
                           CNST FIXED DEC (2) INITIAL (18),  /* LENGTH OF FIXED
                                                              PORTION OF RECORD    */
                           NUMC FIXED (2) INITIAL (0),  /* NUM OF CHAR IN WORD  */
                           VOLU FIXED (2) INITIAL (1),  /* VOLUME NUMBER        */
                           CHAP FIXED (3) INITIAL (1),  /* CHAPTER NUMBER       */
                           PARA FIXED (3) INITIAL (1),  /* PARAGRAPH NUMBER     */
                           SENT FIXED (5) INITIAL (0),  /* SENTENCE NUMBER      */
                           WORD FIXED (3) INITIAL (1),  /* WORD NUMBER          */
                           OTHER CHAR (58) STATIC,
                           AWORD CHAR (58) VARYING;  /* LENGTH OF LONGEST WORD    */
                                                     /* TO BE PROCESSED.          */
6                  DECLARE FIELD  CHAR (82) INITIAL (' ');

              /*********************************************************************/
               /*                                                                  */
              /*  THE PROGRAM REQUIRES THAT THE TYPE OF DATA TO BE PROCESSED BE    */
              /*  SPECIFIED AS THE FIRST FOUR POSITIONS WITHIN THE 'PARM' FIELD    */
              /*  OF  THE JCL EXEC CARD OR THE FIRST INPUT RECORD OF SYSIN.  IF    */
              /*  THE TYPE PARAMETER IS NOT SPECIFIED THE PROGRAM TERMINATES WITH  */
```

114

```
/*  A MSG ON SYSPRINT TO THE USER.  IF THE USER WISHES THE DATA BEING*/
/*  PROCESSED TO BE PRINTED, THE TYPE PARAMETER ON JCL OR SYSIN      */
/*  SHOULD BE FOLLOWED BY A COMMA (,) AND THE PARAMETER 'PRINT' IN   */
/*  THE FORMAT 'XXXX,PRINT'.  IF A PRINTOUT IS NOT REQUIRED THE      */
/*  PARAMETER 'NOPRT' SHOULD BE SPECIFIED USING THE SAME FORMAT.     */
/*  IF NO PRINT CONTROL PARAMETER IS SPECIFIED, THE DEFAULT IS       */
/*  'NOPRT'.                                                         */
/*                                                                   */
/*  TO USE THE FIRST RECORD OF SYSIN TO INTRODUCE PARAMETERS,        */
/*  THE JCL EXEC PARM FIELD MUST CONTAIN THE VALUE 'FRST'.           */
/*    EXAMPLE USING JCL FOR PARM PASSING;                            */
/*       //   EXEC  PL1,PARM.GO='POET,PRINT'                        */
/*                                                                   */
/*  IF A PLAY HAS STAGE DIRECTIONS THEY WILL BE IGNORED UNLESS       */
/*  THE PARAMETER 'STAGE DIRECTION' IS SPECIFIED.  PARM FIELD FORMAT */
/*  IS:                                                              */
/*            'XXXX,XXXXX,STAGE DIRECTION'                           */
/*                                                                   */
/*    EXAMPLE USING FIRST RECORD OF SYSIN FOR PARAMETERS;            */
/*       //    EXEC  PL1,PARM.GO='FRST'                            */
/*    FIRST CARD OF INPUT ON SYSIN WOULD THEN CONTAIN BEGINING IN    */
/*  COLUMN 1,  POET,PRINT                                            */
/*                                                                   */
/*  THE FOLLOWING PL1 CODE TESTS THE VALUE PASSED THROUGH JCL        */
/*  IN THE 'PARM' FIELD.  IF THE VALUE = 'FRST' THE FIRST RECORD     */
/*  OF THE INPUT FILE ON SYSIN CONTAINS THE PARAMETERS IN POSITIONS  */
/*  1 THROUGH 10 AND THESE PARAMETERS ARE USED BY THE PROGRAM BY     */
/*  CONTINUING THE PROGRAM AT LABEL 'FIRST'.  IF THE VALUE IN THE    */
/*  'PARM' FIELD IS SOMETHING OTHER THAN 'FRST' THAT VALUE IS USED   */
/*  BY SETTING 'UHLAB' = TO THE VALUE OF THE FIRST FOUR POSITIONS    */
/*  AND PRNTCTRL = THE VALUE OF POSITIONS 6-10 AND CONTINUING THE    */
/*  PROGRAM AT LABEL 'SECOND'.                                       */
/*                                                                   */
 /*                                                                 */
/*******************************************************************/

          OPEN FILE(SYSIN) INPUT, FILE(SYSPRINT) OUTPUT;
          ON ENDFILE(SYSIN) GO TO OUTOFCRDS;

        IF PARM = 'FRST' THEN GO TO FIRST;
        ELSE  TEMP = PARM;
              UHLAB = SUBSTR(TEMP,1,4);
              PRNTCTRL = SUBSTR(TEMP,6,5);
              IF UHLAB = 'PLAY' | UHLAB = 'HAML' THEN
                STAGEDIRECT = SUBSTR(TEMP,12,15);
              GO TO SECOND;

FIRST:  GET FILE(SYSIN) EDIT (UHLAB,PRNTCTRL,STAGEDIRECT) (A(4),X(1),
          A(5),X(1),A(15),X(54));
SECOND:   IF PRNTCTRL = '     ' THEN PRNTCTRL = 'NOPRT'; /*DEFAULT*/
          IF UHLAB = '    ' THEN DO;  /*  DATA TYPE TO BE PROCESSED  */
                                      /* MUST BE SPECIFIED          */
```

```
23                      PUT FILE(SYSPRINT) EDIT ('TYPE OF DATA TO BE PROCESSED ',
                            'HAS NOT BEEN SPECIFIED.  PROGRAM TERMINATED.')
                            (PAGE,A(29),A(46));
24                      GO TO CLOSING;
25                      END;

26                PUT FILE(SYSPRINT) EDIT ('TYPE OF DATA TO BE PROCESSED IS ',
                        UHLAB) (PAGE,A(32),A(4));
27                PUT FILE(SYSPRINT) EDIT (PRNTCTRL,' HAS BEEN SPECIFIED AS ',
                        'PRINT CONTROL.','    ') (SKIP,A(5),A(23),A(14),
                        SKIP(2),A(4));

28                IF UHLAB = 'PLAY' & STAGEDIRECT = 'STAGE DIRECTION' THEN
29                      PUT FILE(SYSPRINT) EDIT ('PROCESSING OF STAGE DIRECTIONS',
                          ' HAS BEEN SPECIFIED.') (SKIP,A,A);


             /* UHLAB EQUAL TO 'PROS', 'POET', 'PLAY', OR 'SPOK' WILL USE 'COL'  */
             /* INITIAL VALUE OF 73 AS SHOWN ON THE DECLARE STATEMENT FOR 'COL'. */


30                IF UHLAB = 'PROS' THEN DO;
32                      SENT = 1;
33                      TOSE = 1;
34                      GO TO PWA;    /*  PRINT SECTION HEADINGS              */
35                      END;

36                IF UHLAB = 'PLAY' | UHLAB = 'SPOK' THEN GO TO PWA; /* PRINT
                                                 INITIAL SECTION HEADINGS.    */

38                IF UHLAB = 'OPLA' THEN COLIM = 68;

40              L1: CALL FORM;    /*  SUBROUTINE TO DISTINGUISH WORDS          */

            /* RESET COUNTERS BASED ON KEYWORD WITHIN TEXT.                    */

41                IF UHLAB = 'PROS' THEN DO;
43                      IF AWORD = '$VOLUME'     THEN DO; L = L7; GO TO L6; END;
49                      IF AWORD = '$CHAPTER'    THEN DO; L = L8; GO TO L6; END;
53                      IF AWORD = '$PARAGRAPH'  THEN DO; L = L9; GO TO L6; END;
58                      IF AWORD = '$SENTENCE'   THEN DO; L = L10; GO TO L6; END;
63                      GO TO L2;
64                      END;

65                IF UHLAB = 'PLAY' | UHLAB = 'OPLA' THEN DO;
67                      IF SUBSTR(AWORD,1,1) = '*' &
68                          STAGEDIRECT ¬= 'STAGE DIRECTION' THEN GO TO L1;
69                      IF AWORD = '$ACT'        THEN DO; L = L8; GO TO L6; END;
74                      IF AWORD = '$SCENE'      THEN DO; L = L9; GO TO L6; END;
79                      GO TO L2;
80                      END;

81                IF UHLAB = 'SPOK' THEN DO;
```

```
                                                                        116
                    IF AWORD = '$SERIES'    THEN DO; L = L7; GO TO L6; END;
                    IF AWORD = '$SESSION'   THEN DO; L = L8; GO TO L6; END;
                    IF AWORD = '$SPEAKER'   THEN DO; L = L9; GO TO L6; END;
             /*     GO TO L2;      DEFAULT                                  */
                    END;

             /*                                                            */
             /*  THE FOLLOWING TRANSFORMS PARAGRAPH, CHAPTER, AND VOLUME END */
             /* INDICATIONS TO A SINGLE PERIOD '.'.                        */
             /*                                                            */

               L2:  OTHER = '                    ';   /* CLEAR AREA */
                    IF AWORD = '..' | AWORD = '$$$' | AWORD = '$$$$' THEN DO;
                         SUBSTR(OTHER,1,1) = '.';
                         NUMC = 1;
                         END;
                    ELSE SUBSTR(OTHER,1,NUMC)=AWORD;
                         TOWO = TOWO + 1;

             /*  OUTPUTTING RECORD;  RECORD FORMAT IS:                     */
             /*                                                            */
             /*      POSITIONS 01-04 VARIABLE LENGTH COUNT FIELD.          */
             /*      POSITIONS 05-06 LENGTH OF VARIABLE TEXT WORD.         */
             /*      POSITIONS 07-08 VOLUME NUMBER.                        */
             /*      POSITIONS 09-11 CHAPTER NUMBER.                       */
             /*      POSITIONS 12-14 PARAGRAPH NUMBER.                     */
             /*      POSITIONS 15-19 SENTENCE NUMBER.                      */
             /*      POSITIONS 20-22 WORD NUMBER WITHIN SENTENCE.          */
             /*      POSITIONS 23-80 TEXT WORD, VARYING IN LENGTH UP TO 58 CHAR. */

                    PUT STRING(FIELD) EDIT (CNST,'    ',NUMC,VOLU,CHAP,PARA,SENT,
                      WORD,OTHER) (F(2),A(4),F(2),F(2),F(3),F(3),F(5),F(3),
                      A(NUMC));
                    CALL PUTOUT (FIELD);
             /*          DCB=(RECFM=VB,LRECL=80,BLKSIZE=3520)              */
             /*          DDNAME TO BE USED = OUTPUT                        */


              /*  AT THIS POINT, UHLAB EQUAL TO 'PROS' WILL                 */
              /*  CAUSE THE VARIOUS COUNTERS TO BE INCREMENTED UNDER THE FOLLOWING*/
              /* CONDITIONS AND/OR THE PROGRAM TO BRANCH TO THE LABEL 'PW'.  ALL */
              /*  OTHER UHLAB VALUES RESULT IN A BRANCH TO PW.              */

                   IF UHLAB = 'PROS' THEN DO;
                         IF AWORD = '.' THEN GO TO CS;     /* SENTENCE     */
                         IF AWORD = '?' THEN GO TO CS;     /*  SENTENCE    */
                         IF AWORD = '!' THEN GO TO CS;     /* SENTENCE     */
                         END;

                   IF UHLAB = 'PROS' | UHLAB = 'PLAY' | UHLAB = 'SPOK'
                     THEN DO;
                         IF AWORD = '..' THEN GO TO CP;    /*  PARAGRAPH   */
                         IF AWORD = '$$$'  THEN GO TO CC;  /* CHAPTER      */
```

```
124                         IF AWORD = '$$$$' THEN GO TO CV;  /*  VOLUME          */
126                         END;

127                   GO TO PW;

128                   CV:   TOVO = TOVO+1;
129                         VOLU = VOLU+1;
130                         CHAP = 0;
131                   CC:   TOCH = TOCH+1;
132                         CHAP = CHAP+1;
133                         PARA = 0;
134                   CP:   TOPA = TOPA+1;
135                         PARA = PARA+1;
136                         SENT = 0;
137                   CS:   TOSE = TOSE+1;
138                         SENT = SENT+1;
139                         WORD = 0;
140                   PW:   WORD = WORD+1;
141                         IF PRNTCTRL = 'NOPRT' THEN GO TO L1;

143                     IF UHLAB = 'PROS' | UHLAB = 'PLAY' | UHLAB = 'SPOK'
144                       THEN DO;
145                         PUT FILE(SYSPRINT) EDIT (AWORD,' ') (A(NUMC), A(1));
146                         IF AWORD = '..' | AWORD = '$$$' | AWORD = '$$$$'
147                             THEN DO;
148                   PWA:  IF PRNTCTPL = 'NOPRT' THEN GO TO L1;

150                   PUT FILE(SYSPRINT) EDIT ('VOLU = ','SERIES = ',VOLU,
                            '  CHAP = ','ACT = ','ACT = ','  SESSION = ',CHAP,
                            '  PARA = ','  SCENE = ','  SCENE = ','  SPEAKER = ',
                      PARA,' ') (SKIP(2),A((UHLAB='PROS')*7),
                                         A((UHLAB='SPOK')*9),F(2),
                  /* FORMATS SECTION      */   A((UHLAB='PROS')*9),
                  /* HEADING BASED ON TYPE*/   A((UHLAB='PLAY')*6),
                  /* OF PROCESSING        */   A((UHLAB='HAML')*6),
                  /* SPECIFIED IN FIELD   */   A((UHLAB='SPOK')*12),F(3),
                  /* 'UHLAB'.             */   A((UHLAB='PROS')*9),
                                                A((UHLAB='PLAY')*10),
                                                A((UHLAB='HAML')*10),
                                                A((UHLAB='SPOK')*12),F(3),
                                                A(1));
151                         GO TO L1;  END;

153                         END;
154                     GO TO L1;

155                   L6:   PUT FILE(SYSPRINT) EDIT ((CARDIMAGE(I) DO I = 1 TO 72))
156                             (SKIP(2),72 A(1));  PUT SKIP;
157                         CALL FORM;
158                         COL = 73;     /* FORCE READING OF NEW RECORD          */
159                         GO TO L;

160                   L7:   VOLU = AWORD;
```

```
161                         CHAP = 1;  PARA = 1;  SENT = 1;  WORD = 1;
165                         GO TO PWA;
166                   L8:   CHAP = AWORD;
167                         PARA = 1;  SENT = 1;  WORD = 1;
170                         GO TO PWA;
171                   L9:   PARA = AWORD;
172                         SENT = 1;  WORD = 1;
174                         GO TO PWA;
175                   L10:  SENT = AWORD;
176                         WORD = 1;
177                         GO TO PWA;

178                   OUTOFCRDS: PUT FILE(SYSPRINT) EDIT ('NUMBER OF WORDS ',
                            '(CONTENT AND FUNCTION) = ',TOWO)      (PAGE,A(16),A(25),
                            F(6));

179                     IF UHLAB = 'PROS' THEN DO;
181                     PUT FILE(SYSPRINT) EDIT ('NUMBER OF SENTENCES = ',TOSE)
                                (SKIP,A(22),F(6));
182                     PUT FILE(SYSPRINT) EDIT ('NUMBER OF PARAGRAPHS =',TOPA)
                            (SKIP,A(22),F(6));
183                     PUT FILE(SYSPRINT) EDIT ('NUMBER OF CHAPTERS =     ',TOCH)
                            (SKIP,A(24),F(4));
184                     PUT FILE(SYSPRINT) EDIT ('NUMBER OF VOLUMES =      ',TOVO)
                            (SKIP,A(26),F(2));
185                     GO TO CLOSING;
186                     END;

187                     IF UHLAB = 'PLAY' | UHLAB = 'HAML' THEN DO;
189                     PUT FILE(SYSPRINT) EDIT ('NUMBER OF SENTENCES = ',TOSE)
                                (SKIP,A(22),F(6));
190                     PUT FILE(SYSPRINT) EDIT ('NUMBER OF SCENES = ',TOPA)
                                (SKIP,A(19),F(6));
191                     PUT FILE(SYSPRINT) EDIT ('NUMBER OF ACTS = ',TOCH)
                                (SKIP,A(17),F(4));
192                     GO TO CLOSING;
193                     END;

194                     IF UHLAB = 'SPOK' THEN DO;
196                     PUT FILE(SYSPRINT) EDIT ('NUMBER OF SENTENCES = ',TOSE)
                                (SKIP,A(22),F(6));
197                     PUT FILE(SYSPRINT) EDIT ('NUMBER OF SPEAKERS = ',TOPA)
                                (SKIP,A(21),F(6));
198                     PUT FILE(SYSPRINT) EDIT ('NUMBER OF SESSIONS = ',TOCH)
                                (SKIP,A(21),F(4));
199                     PUT FILE(SYSPRINT) EDIT ('NUMBER OF SERIES = ',TOVO)
                                (SKIP,A(19),F(2));
200                     GO TO CLOSING;
201                     END;

202                         PUT FILE(SYSPRINT) EDIT ('NUMBER OF INPUT RECORDS = ',
                              TOSE) (SKIP,A(25),F(6));
```

```
203                   CLOSING:  CLOSE FILE(SYSIN),              FILE(SYSPRINT);
204                     CALL CLSEOUT;  /* CLOSES INDX FILE.                    */
205                   GO TO FNDINDX;

                  /*************************************************************************/
                   /*                                                                       */
                   /*   SUBROUTINE 'FORM'                                                   */
                   /*                                                                       */
206                     FORM: PROCEDURE;
207                           NUMC = 0;
208                           AWORD = '';
209                     BUMP: COL = COL+1;
210                           IF COL < COLIM THEN GO TO EXTRACT;
212                           IF COL = COLIM THEN DO;
214                              IF CARDIMAGE(COL) = ' ' THEN GO TO LSTCHAR;
216                              IF CARDIMAGE(COL) = '-' THEN DO;  /* WORD CONTINUED */
218                                 WRDCONT = 1;
219                                 GO TO BUMP;
220                                 END;
221                              IF CARDIMAGE(COL) = '@' THEN DO;  /* SENTENCE CONT  */
223                                 LINECONT = 1;
224                                 GO TO BUMP;
225                                 END;
226                              IF CARDIMAGE(COL) = '#' THEN DO;  /* WORD AND       */
228                                 WRDCONT = 1;                   /* SENTENCE CONT  */
229                                 LINECONT = 1;
230                                 GO TO BUMP;
231                                 END;
232                              END;
233                           COL = 1;
234                           IF UHLAB = 'OPLA' THEN GO TO READ68;

236                     READ:   GET FILE(SYSIN) EDIT ((CARDIMAGE(I) DO I = 1 TO 72),
                                        LINEA,SEQUENCE) (72 A(1),F(2),A(6));

237                           IF UHLAB = 'POET' | UHLAB = 'MILT' THEN DO;
239                             IF WRDCONT = 0 THEN WORD = 1;
241                              WRDCONT = 0;
242                             IF LINECONT = 0 THEN DO;
244                                SENT = SENT + 1;
245                                TOSE = TOSE + 1;
246                                END;
247                              LINECONT = 0;
249                             IF UHLAB = 'MILT' THEN VOLU = LINEA;
250                             IF PRNTCTRL = 'NOPRT' THEN GO TO EXTRACT;
252                             PUT FILE(SYSPRINT) EDIT ((CARDIMAGE(I) DO I = 1 TO 72),
                                   VOLU,'-',SENT) (SKIP,COLUMN(10),72 A(1),X(2),
                                   F((UHLAB='MILT')*2),A(UHLAB='MILT'),F(5));
253                             GO TO EXTRACT;
254                           END;

255                           IF UHLAB = 'SPOK' THEN SENT = SENT + 1;
```

```
257             EXTRACT:
258                         IF CARDIMAGE(COL) = ' ' THEN GO TO LSTCHAR;
259                         NUMC = NUMC+1;
260                         AWORD = AWORD || CARDIMAGE(COL);  /*CONCATENATION  */
261                         GO TO BUMP;

262             READ68:
                            GET FILE(SYSIN) EDIT ((CARDIMAGE(I) DO I = 1 TO 68),
                                  LINE,SEQUENCE) (68 A(1),F(4),A(8));
263                         IF SENT ¬= LINE THEN DO;
265                             SENT = LINE;
266                             TOSE = TOSE + 1;
267                             WORD = 1;
268                             END;
269                         IF PRNTCTRL = 'NOPRT' THEN GO TO EXTRACT;
271                         PUT FILE(SYSPRINT) EDIT ((CARDIMAGE(I) DO I = 1 TO 68),
                                  SENT,SEQUENCE) (SKIP,COLUMN(10),68 A(1),X(2),
                                  F(5),X(2),A(8));
272                         GO TO EXTRACT;

273             LSTCHAR:
274                         IF NUMC = 0 THEN GO TO BUMP;
275                         RETURN;

276                         END;

                 /*   END OF SUBROUTINE 'FORM'                                  */

                /*************************************************************************/


277             ENDINDX:  END INDEX;
```

SUFUN Program Listing

```
1               SUFUN:  PROC  OPTIONS(MAIN);

               /*  THE PURPOSE OF THIS PROGRAM IS TO PRODUCE A DATA SET WHICH       */
               /*  CONTAINS LINKED PAIRS OF SUFFIXES AND THEIR POSSIBLE EXCEPTIONS  */
               /*  FOLLOWED BY FUNCTION WORDS.  THE OUTPUT DATA SET WILL BE USED BY */
               /*  THE PROGRAM 'SUFFIX' TO LINK CONTENT WORDS BY ROOT FORM.         */

2                     DECLARE
                        CARDIMAGE (80) CHAR (1);       /* CARD READ IN AREA          */

3                     DECLARE
                        COL FIXED DEC (2),             /* COLUMN POINTER             */
                        J1 BINARY FIXED (15,0),
                        I   BINARY FIXED (15,0);

4                     DECLARE
                        NSUF1 FIXED DEC (3) INITIAL (0),  /* NUMBER OF SUFFIXS ONE   */
                        NSUFX FIXED DEC (3) INITIAL (0),  /* NUMBER OF SUFFIXS 2     */
                        NFUN FIXED DEC (3) INITIAL (0),   /* NUMBER OF FUNCTION WDS  */
                        NEXCP FIXED DEC (4) INITIAL (1);  /* NUMBER OF EXCEPTIONS WDS*/

5                     DECLARE
                        1 WORK1,
                          2 NUMC FIXED DEC (1),
                          2 LOC FIXED DEC (4) INITIAL (0), /*PRESET LOCEXC & LOCSUF2 */
                          2 AWORD CHAR (18) VARYING;

6                     DECLARE
                        1 WORK2,
                          2 TEMPLEN FIXED DEC (2),
                          2 LWORD CHAR (18) VARYING;

7                     DECLARE
                        1 SUFFIX1 (500),
                          2 LENSUF1 FIXED DEC (1),     /* LENGTH OF WORD SUF1.       */
                          2 LOCSUF2 FIXED DEC (3),     /* LOCATION OF SUF2 LIST      */
                          2 SUF1 CHAR (8) VARYING;     /* SUFFIX ONE OF PAIR.        */

8                     DECLARE
                        1 SUFFIX2 (500),
                          2 LENSUF2 FIXED DEC (1),     /* LENGTH OF WORD SUF2.       */
                          2 LOCEXC FIXED DEC (4),      /* LOCATION OF EXCEPTION LIST */
                          2 SUF2 CHAR (8) VARYING;     /* SUFFIX 2 OF SUFFIX PAIR    */

9                     DECLARE
                        1 EXCEPTWD (1000),
                          2 LENEXC FIXED DEC (2),      /* LENGTH OF EXCEPT WORD      */
                          2 EXCPT CHAR (18) VARYING;   /* EXCEPTION WORD.            */

10                    DECLARE
                        1 FUNCTIONWD (200),
                          2 LENFUN FIXED DEC (2),      /* LENGTH OF FUNCTION WORD    */
                          2 FUN CHAR (18) VARYING;     /* FUNCTION WORD              */
```

```
11                    OPEN FILE(SYSIN) INPUT,                                  0047
                           FILE(SYSPRINT) OUTPUT,                              0048
                           FILE(SUF) OUTPUT;                                   0049
12                    ON ENDFILE(SYSIN) GO TO SORTFUN;                         0050

14            RW1:    COL = 73 ;    /* FORCE READING OF NEW INPUT RECORD      */
15                    CALL FORM;                                               0053
16                    IF AWORD = 'EXCEPT' THEN GO TO EXCEPTION;                0054
19                    IF LWORD = 'EXCEPT' THEN DO;                             0055
20                       EXCPT(NEXCP) = '0';                                   0056
21                       LENEXC(NEXCP) = 1;                                    0057
22                       NEXCP = NEXCP+1;                                      0058
23                       IF NEXCP > 1000 THEN GOTO FLOVER1;  /* OVERFLOW HAS OCCURR*/
25                       END;                                                  0060
26                    IF AWORD = 'FUNCTIONWORDS' THEN GO TO SORTSUF;           0061
28                    NSUFX = NSUFX+1;                                         0062
29                    IF NSUFX > 500 THEN GO TO FLOVER2;  /* OVERFLOW HAS OCCURRED */
31                    SUFFIX1(NSUFX) = WORK1; /* PRESETS LOCSUF2 TO ZERO     */
32                    LWORD = AWORD;                                           0066
33                    CALL FORM;                                               0067
34                    SUFFIX2(NSUFX) = WORK1; /* PRESETS LOCEXC TO ZERO      */
35                    GO TO RW1;                                               0071

36            EXCEPTION:                                                       0072
37                    IF LWORD = 'EXCEPT' THEN GOTO REXCEPT;                   0073
38                      LWORD = 'EXCEPT';                                      0074
39                      LOCEXC(NSUFX) = NEXCP;                                 0075

40            REXCEPT:                                                         0076
                      CALL FORM;                                               0077
41                    IF AWORD = 'LETTER' THEN DO;                             0078
43                       CALL FORM;                                            0079
44                       EXCPT(NEXCP)='-'||AWORD;                              0080
45                       LENEXC(NEXCP)=2;                                      0081
46                       GO TO INCRE;
47                       END;                                                  0083
48                    EXCPT(NEXCP) = AWORD;                                    0084
49                    LENEXC(NEXCP) = NUMC;                                    0085

50            INCRE:
                      NEXCP = NEXCP+1;                                         0087
51                    IF NEXCP > 1000 THEN GO TO FLOVER3;  /* OVERFLOW HAS OCCURRED*/
53                    GOTO RW1;                                                0089


              /*  THE FOLLOWING CODE INTERCHANGES SUFFIXES OF A SUFFIX PAIR SO THAT
                     THE LOWER ORDER SUFFIX IS PLACED IN 'SUFFIX1'.           */

54            SORTSUF:
                      LOCEXC(NSUFX+1) = NEXCP;                                 0093
55                    EXCPT(NEXCP) = '0';                                      0094
```

```
56                    LENEXC(NEXCP) = 1;                                    0095
57                    DO I = 1 TO NSUFX;
58                      IF SUF1(I) <= SUF2(I) THEN GO TO N1;
60                      WORK1 = SUFFIX1(I);
61                      SUFFIX1(I) = SUFFIX2(I);
62                      SUFFIX2(I) = WORK1;
63                N1:   END;                                                  0100

                    /* SORT SUFFIX1:
                          SORT SUFFIXS APPEARING IN SUF1 IN ASCENDING SEQUENCE MAINTAINING
                            THE RELATIONSHIP WITH THE DIRECT OPPOSITE OF THE PAIR IN
                            SUFFIX2.                                        */

64                    DO I = 1 TO NSUFX-1;                                  0102
65                       IF SUF1(I) > SUF1(I+1) THEN DO;                    0103
67                          DO J = I+1 TO 2 BY -1 WHILE (SUF1(J) < SUF1(J-1));  0104
68                            WORK1 = SUFFIX1(J);
69                            SUFFIX1(J) = SUFFIX1(J-1);
70                            SUFFIX1(J-1) = WORK1;

71                            WORK1 = SUFFIX2(J);
72                            SUFFIX2(J) = SUFFIX2(J-1);
73                            SUFFIX2(J-1) = WORK1;
74                          END;   /* J */                                  0110
75                      END;       /* SWAP */                              0111
76                    END;         /* I & SORT */                          0112

                    /*    ELIMINATE DUPLICATES IN SUF1.  THE FIELD 'LOCSUF2' WILL CONTAIN
                            A POINTER TO THE FIRST SUFFIX IN SUF2 WHEN THERE ARE MULTIPLE
                            DUPLICATE ENTRIES IN SUF1.                      */

77                    J1 = 2;                                               0115
78                    LOCSUF2(1) = 1;                                       0116
79                    DO I = 1 TO NSUFX-1;
80                    IF SUF1(I) = SUF1(I+1) THEN GO TO N3;                 0118
82                       LENSUF1(J1) = LENSUF1(I+1);                        0119
83                       SUF1(J1) = SUF1(I+1);                              0120
84                       LOCSUF2(J1) = I+1;                                 0121
85                       J1 = J1+1;                                         0122
86                N3:   END;                                                0123

87                    LOCSUF2(J1) = NSUFX+1;   /* TO TERMINATE LAST SUF2 */ 0124
88                    NSUF1 = J1-1;                                         0125

                    /* SORT SUFFIX 2:
                          SORT SUFFIXS IN SUFFIX2 WITHIN COMMON SUFFIX1.  GROUP WILL BE
                          SORTED ONLY WHERE A DUPLICATE OF A SUFFIX APPEARS IN SUFFIX1;  */

89                    DO I = 1 TO NSUF1-1;
90                    IF LOCSUF2(I) = LOCSUF2(I+1)-1 THEN GO TO N5;         0129
92                       DO J = LOCSUF2(I) TO LOCSUF2(I+1)-2;               0130
93                          IF SUF2(J) > SUF2(J+1) THEN DO;                 0131
95                             DO K = J+1 TO LOCSUF2(I)+1 BY -1             0132
```

```
                          WHILE (SUF2(K) < SUF2(K-1));                    0133
 96                      WORK1 = SUFFIX2(K);
 97                      SUFFIX2(K) = SUFFIX2(K-1);
 98                      SUFFIX2(K-1) = WORK1;
 99                    END;  /* K */                                       0137
100                 END;     /* SWAP */                                    0138
101               END;       /* J GROUP */                                 0139
102     N5:     END;         /* I & SORT */                                0140

        /* SORT EXCEPT LIST:
              SORT EXCEPT WORDS MAINTAINING RELATIONSHIP TO SUF1 AND SUF2.   */

103       J1 = 1;                                                           0143
104     N6: DO J = J1 TO NEXCP;                                             0144
105           IF EXCPT(J+1) = '0' THEN DO;                                  0145
107             J1 = J+2;                                                   0146
108             IF J1 >= NEXCP THEN GOTO OUTPUT;                            0147
110             GO TO N6;                                                   0148
111             END;           /* LIST */                                   0149
112           IF EXCPT(J) > EXCPT(J+1) THEN DO;                             0150
114             DO K = J+1 TO J1+1 BY -1 WHILE (EXCPT(K)<EXCPT(K-1));       0151
115                WORK2 = EXCEPTWD(K);
116                EXCEPTWD(K) = EXCEPTWD(K-1);
117                EXCEPTWD(K-1) = WORK2;
118             END;           /* K */                                      0154
119           END;             /* SWAP */                                   0155
120         END;         /* J GROUP */                                      0156

        /* OUTPUT SUFFIX RECORDS.  RECORDS ARE PADDED TO STANDARDIZE RECORD
            LENGTH TO 20 BYTES.                                           */

121     OUTPUT:                                                             0158
              PUT FILE(SUF) EDIT (NSUF1,NSUFX,NEXCP,'          ')
                    (F(3),F(3),F(4),A(10));

122           DO I = 1 TO NSUF1;
123             PUT FILE(SUF) EDIT (SUFFIX1(I),'                ')
                      (F(1),F(3),A(LENSUF1(I)),A(16-LENSUF1(I)));

124               END;                                                      0170
125           DO I = 1 TO NSUFX;
126             PUT FILE(SUF) EDIT (SUFFIX2(I),'                ')
                      (F(1),F(4),A(LENSUF2(I)),A(15-LENSUF2(I)));
127             END;                                                        0164

128           DO I = 1 TO NEXCP;
129             PUT FILE(SUF) EDIT (EXCEPTWD(I),'                  ')
                      (F(2),A(LENEXC(I)),A(18-LENEXC(I)));
130             END;                                                        0174

        /* PRINT OUT SUFFIX AND EXCEPTION WORDS.                         */

131     HEAD1:
```

```
                PUT FILE(SYSPRINT) EDIT ('SUFFIX PAIRS AND ASSOCIATED EXCEPTION',
                  ' WORDS','NUMBER','NUMBER','NUMBER','EXCEPTION','SUFFIX 1',
                  'SUFFIX 2','WORDS',NSUF1,NSUFX,NEXCP) (PAGE,A,A,SKIP(2),
                  COLUMN(23),A,SKIP,X(1),A,X(4),A,X(3),A,SKIP,A,X(2),A,X(4),A,
                  SKIP,X(2),F(3),X(7),F(3),X(7),F(4));
132             PUT FILE(SYSPRINT) EDIT ('EXCEPTION','INDEX','INDEX','SUFFIX 1',
                  'SUFFIX 2','WORDS','SUFFIX 2','EXCEPTION') (SKIP(2),COLUMN(21),
                  A,COLUMN(42),A,COLUMN(53),A,SKIP,A,X(2),A,X(4),A,COLUMN(41),
                  A,X(2),A);
133             PUT FILE(SYSPRINT) SKIP(2);

134           PRTDATA:
                DO I = 1 TO NSUF1;
135               ON ENDPAGE(SYSPRINT) BEGIN;
137               PUT FILE(SYSPRINT) EDIT ('EXCEPTION','INDEX','INDEX',
                    'SUFFIX 1','SUFFIX 2','WORDS','SUFFIX 2','EXCEPTION')
                    (PAGE,COLUMN(21),A,COLUMN(42),A,COLUMN(53),A,SKIP,A,X(2),
                    A,X(4),A,COLUMN(41),A,X(2),A);
138               PUT FILE(SYSPRINT) SKIP(2);
139               END;

140             PUT FILE(SYSPRINT) EDIT (SUF1(I),LOCSUF2(I))
                  (SKIP,A(LENSUF1(I)),COLUMN(43),F(3));

141             DO J = LOCSUF2(I) TO LOCSUF2(I+1) - 1;
142               PUT FILE(SYSPRINT) EDIT (SUF2(J),LOCEXC(J))
                    (SKIP(0),COLUMN(11),A(LENSUF2(J)),COLUMN(53),F(4));
143               IF LOCEXC(J) ¬= 0 THEN DO;
145                 DO K = LOCEXC(J) BY 1 WHILE (EXCPT(K) ¬= '0');
146                   PUT FILE(SYSPRINT) EDIT (EXCPT(K)) (SKIP,COLUMN(21),
                        A(LENEXC(K)));
147                   END;  /* DO LOOP K    */
148                 END;   /* IF STATEMENT     */
149                 PUT FILE(SYSPRINT) SKIP;
150                 END;  /* DO LOOP  J   */
151             END;  /* DO LOOP I  */


              /* BEGIN PROCESSING OF FUNCTION WORDS.                          */

152           RF:     COL = 73;            /* NEW CARD */                      0177
153                   CALL FORM;                                               0178
154                   NFUN = NFUN+1;                                           0179
155                   IF NFUN > 200 THEN GO TO FLOVER4;
157                   FUN(NFUN) = AWORD;                                       0181
159                   LENFUN(NFUN) = NUMC;                                     0182
159                   GOTO RF;                                                 0183

              /* SORT FUNCTION WORDS IN ASCENDING SEQUENCE.                   */

160           SORTFUN:
                DO I = 1 TO NFUN-1;                                           0186
161                IF FUN(I) > FUN(I+1) THEN DO;                              0187
```

```
                                                                                127
163                     DO J = I+1 TO 2 BY -1 WHILE (FUN(J) < FUN(J-1));          0188
164                       WORK2 = FUNCTIONWD(J);
165                       FUNCTIONWD(J) = FUNCTIONWD(J-1);
166                       FUNCTIONWD(J-1) = WORK2;
167                     END;  /* J */                                             0191
168                   END;    /* SWAP */                                          0192
169                 END;      /* I & SORT */                                      0193

              /* OUTPUT FUNCTION RECORDS AND PRINT FUNCTION WORDS.                */

170           HEAD2:
                PUT FILE(SYSPRINT) EDIT ('FUNCTION WORDS  -  NUMBER OF ',
                  'FUNCTION WORDS PROCESSED = ',NFUN) (PAGE,A,A,F(3));
171             PUT FILE(SYSPRINT) EDIT ('INDEX  FUNCTION WORD')(SKIP(2),X(2),A);
172             PUT FILE(SYSPRINT) SKIP(2);

173             PUT FILE(SUF) EDIT (NFUN,'                  ') (F(3),A(17));


174           PRTDATA2:
                DO I = 1 TO NFUN;
175               PUT FILE(SUF) EDIT (FUNCTIONWD(I),'                    ')
                      (F(2),A(LENFUN(I)),A(18-LENFUN(I)));
176               ON ENDPAGE(SYSPRINT) BEGIN;
178                 PUT FILE(SYSPRINT) EDIT ('INDEX  FUNCTION WORD')
                      (PAGE,X(2),A);
179                 PUT FILE(SYSPRINT) SKIP(2);
180                 END;
181               PUT FILE(SYSPRINT) EDIT (I, FUN(I)) (SKIP,X(3),F(3),X(4),
                    A(LENFUN(I)));
182               END;

183               GO TO CS;

              /* OVERFLOW ERROR MESSAGES                                          */

184           FLOVER1:
                PUT FILE(SYSPRINT) EDIT ('*** OVER FLOW HAS OCCURRED.  MORE ',
                  'THAN 500 SUFFIX 1 ARE PRESENT.  PROGRAM TERMINATED.',
                  '  TO CORRECT DECREASE NUMBER OF SUFFIX CARDS OR INCREASE ',
                  'STRUCTURE SIZE OF SUFFIX1 WITHIN PROGRAM AND RESUBMIT.')
                  (SKIP(3),A,A,SKIP,A,A);
185             GO TO CS;

186           FLOVER2:
                PUT FILE(SYSPRINT) EDIT ('*** OVERFLOW HAS OCCURRED.  MORE',
                  ' THAN 500 SUFFIX 2 ARE PRESENT.  PROGRAM TERMINATED.',
                  'TO CORRECT DECREASE NUMBER OF SUFFIX CARDS OF INCREASE ',
                  'STRUCTURE SIZE OF SUFFIX2 WITHIN PROGRAM AND RESUBMIT.')
                  (SKIP(3),A,A,SKIP,A,A);
187             GO TO CS;

188           FLOVER3:
```

```
                PUT FILE(SYSPRINT) EDIT ('** OVER FLOW HAS OCCURRED.  MORE ',
                  'THAN 1000 EXCEPT WORDS ENCOUNTERED.  PROGRAM TERMINATED.',
                  ' TO CORRECT DECREASE NUMBER OF EXCEPTION WORDS.')
                  (SKIP(3),A,A,SKIP,A);
189             GO TO CS;

190         FLOVER4:
                PUT FILE(SYSPRINT) EDIT ('*** OVER FLOW HAS OCCURRED.  MORE ',
                  ' THAN 200 FUNCTION WORDS ENCOUNTERED.  PROGRAM TERMINATED.',
                  'TO CORRECT DECREASE NUMBER OF FUNCTION WORDS.')
                  (SKIP(3),A,A,SKIP,A);

191         CS:     CLOSE FILE(SYSPRINT);                                  0223
192             CLOSE FILE(SUF);                                           0201

193             GO TO ENDSUF;

            /*************************************************************************/
            /*************************************************************************/

            /*  FORM IS A SUBROUTINE TO OBTAIN A SUFFIX OR OTHER WORD FROM A CARD
                  USING BLANKS AS WORD DELIMINATORS.  IF THE FIRST SUFFIX IN A
                  SUFFIX PAIR IS A BLANK CARD COLUMN 1 MUST BE A BLANK.          */

194           FORM: PROCEDURE;                                             0030
195                 NUMC = 0;                                              0031
196                 AWORD = '';                                            0032
197           BUMP: COL = COL+1;                                           0033
198                 IF COL <= 72 THEN GO TO EXTRACT;                       0034
200                 GET FILE(SYSIN) EDIT (CARDIMAGE) (80 A(1));            0035
201                 COL = 1;                                               0036
202                 IF CARDIMAGE(COL) = ' ' THEN DO;
204                   NUMC = 1;          /* NECESSARY FOR 1 BLANK CHAR OUTPUT     */
205                   AWORD = ' ';
206                   RETURN;
207                   END;

208           EXTRACT:                                                     0038
209                   IF CARDIMAGE(COL) = ' ' THEN GO TO LSTCHR;
210                   NUMC = NUMC+1;                                       0041
211                   AWORD = AWORD || CARDIMAGE(COL);                     0042
212                   GO TO BUMP;                                          0043
213           LSTCHR:                                                      0044
214                   IF NUMC = 0 THEN GO TO BUMP;                         0045
215                   RETURN;

216                 END FORM;

            /*************************************************************************/
            /*************************************************************************/

217         ENDSUF: END SUFUN;
```

SUFFIX Program Listing

```
1               SUFFIX: PROC OPTIONS(MAIN);                                    0004

                /*  THE PURPOSE OF THIS PROGRAM IS TO PRODUCE A DATA SET WHICH CONTAINS
                    SINGLE OCCURRENCES OF CONTENT WORDS LINKED BY ROOT FORM TO BE USED
                    BY THE PROGRAM 'THESAUR'.                                   */

2                 DECLARE
                    1 SUFFIX1 (500),
                      2 LENSUF1 FIXED DEC (1),     /* LENGTH OF SUF1 WORD        */
                      2 LOCSUF2 FIXED DEC (3),     /* LOCATION OF SECOND SUFFIX  */
                      2 SUF1 CHAR (8) VARYING;     /* SUFFIX 1 OF SUFFIX PAIR    */

3                 DECLARE
                    1 SUFFIX2 (500),
                      2 LENSUF2 FIXED DEC (1),     /* LENGTH OF SUFFIX 2         */
                      2 LOCEXC FIXED DEC (4), /* LOCATION OF EXCEPT WORD        */
                      2 SUF2 CHAR (8) VARYING;     /* SUFFIX 2 OF SUFFIX PAIR    */

4                 DECLARE
                    1 EXCEPTWD (1000),
                      2 LENEXC FIXED DEC (2), /* LENGTH OF EXCEPT WORD          */
                      2 EXCPT CHAR (18) VARYING;  /* EXCEPTION WORD             */

5                 DECLARE
                    1 FUNCTIONWD (200),
                      2 LENFUN FIXED DEC (2), /* LENGTH OF FUNCTION WORD        */
                      2 FUN CHAR (18) VARYING;     /* FUNCTION  WORD            */

6                 DECLARE
                    1 ROOTCK (1000),
                      2 LENWD FIXED DEC (2),       /* LENGTH OF CONTENT WD       */
                      2 MAT FIXED BINARY (15,0),   /* MATCH COUNT                */
                      2 OCCUR FIXED BINARY (15,0),  /* FREQUENCY OF OCCUR        */
                      2 WD CHAR (58) VARYING;      /* CONTENT WORD              */

7                 DECLARE
                    1 CONTENTWD,
                      2 WORKLGTH FIXED DEC (2),    /* WORK LENGTH               */
                      2 WORKMAT FIXED BINARY (15,0) INITIAL (0),  /* MATCH COUNT */
                      2 WORKFREQ FIXED BINARY (15,0) INITIAL (0), /* FREQ OCCUR */
                      2 WORKWD CHAR (58) VARYING INITIAL (' ');  /* WORD       */

8                 DECLARE
                    NEXCP          FIXED (4),         /* NUMBER OF EXCEPTIONS */    0016
                    NSUFX          FIXED (3),         /* NUMBER OF SUFFIXS */       0017
                    NSUF1          FIXED (3),         /* NUMBER OF SUF1S */         0018
                    NFUN           FIXED (3);         /* NUMBER OF FUNCTION WDS */

9                 DECLARE
                    SWD            CHAR  (3),         /* THREE CHARACTERS OF 64   */
                    SAME_ROOT FIXED DEC (1),      /* BOOLEAN FUNCTION FOR STEM   */
                    MATCNT FIXED BINARY (15,0) INITIAL (1); /* DIFFERENT ROOT CNT*/
```

```
10                  DECLARE
                            TEMP1 CHAR (80),   /*  READ IN WORK AREA.              */
                            OUTAREA CHAR (80), /* OUTPUT WORK AREA.               */
                            CNST FIXED DEC (2) INITIAL (12); /* LENGTH OF INDX INFO*/

11                  DECLARE SEXT FILE SEQUENTIAL RECORD;

12                  DECLARE     /* USED IN PROCEDURE 'CONTENTPROC'                */
                      EOFSW FIXED DEC (1) INITIAL (0), /* END OF FILE SWITCH      */
                      IA FIXED BINARY (15,0) INITIAL (1) STATIC;  /* COUNTER      */

13                  DECLARE   /*  READ IN AREAS, REFERENCED ONLY WITHIN PROCEDURE  */
                              /* 'CONTENTPROC'.                                    */
                      NUMC FIXED DEC (2),          /* WORK LENGTH                 */
                /*    VOLU FIXED DEC (2),         *//* VOLUME NUMBER              */
                /*    CHAP FIXED DEC (3),         *//* CHAPTER NUMBER             */
                /*    PARA FIXED DEC (3),         *//* PARAGRAPH NUMBER           */
                /*    SENT FIXED DEC (5),         *//* SENTENCE NUMBER            */
                /*    WORD FIXED DEC (3),         *//* WORD NUMBER WITHIN SENTENCE */
                      READWD CHAR (58) VARYING;    /* WORD , CAN BE UP TO 58 CHAR */


14                    OPEN FILE(SUF) INPUT;          /* INPUT FROM SUFUN          */

15                    PUT FILE(SYSPRINT) EDIT ('WORDS MATCHED BY ROOT','MATCH',
                            'FREQUENCY','COUNT','WORD','OF OCCURRENCE') (PAGE,A,
                            SKIP(2),COLUMN(3),A,COLUMN(67),A,SKIP,COLUMN(3),A,
                            COLUMN(10),A,COLUMN(65),A);

                  /* READ IN SUFFIX1, SUFFIX2, EXCEPTION WORDS, AND FUNCTION WORD LISTS.
                     'TEMP1' DOES NOT CONTAIN USEFUL DATA FROM RECORDS AT THIS POINT.
                      USED ONLY FOR EXPEDIENCY IN 'GET' STATEMENTS WHICH FOLLOW.      */

16                      GET FILE(SUF) EDIT (NSUF1,NSUFX,NEXCP,TEMP1)
                          (F(3),F(3),F(4),A(10));

17                    DO I = 1 TO NSUF1;
18                      GET FILE(SUF) EDIT (SUFFIX1(I),TEMP1)
                          (F(1),F(3),A(LENSUF1(I)),A(16-LENSUF1(I)));
19                      END;

20
21                    DO I = 1 TO NSUFX;
                        GET FILE(SUF) EDIT (SUFFIX2(I),TEMP1)
                          (F(1),F(4),A(LENSUF2(I)),A(15-LENSUF2(I)));
22                      END;

23                    DO I = 1 TO NEXCP;
24                      GET FILE(SUF) EDIT (EXCEPTWD(I),TEMP1)
                          (F(2),A(LENEXC(I)),A(18-LENEXC(I)));
25                      END;

26                      GET FILE(SUF) EDIT (NFUN,TEMP1)  (F(3),A(17));
```

```
27                    DO I = 1 TO NFUN;
28                      GET FILE(SUF) EDIT (FUNCTIONWD(I),TEMP1)
                          (F(2),A(LENFUN(I)),A(18-LENFUN(I)));
29                      END;

30                    CLOSE FILE(SUF);

31                    OPEN FILE(SEXT) INPUT;   /* INPUT FROM INDEX PROGRAM.        */

                  /*  THE FOLLOWING CODE PROCESSES CONTENT WORDS WITH THE FIRST THREE
                      CHARACTERS EQUAL.  WHEN NOT EQUAL THE CODE WRITES OUT THE
                      CONTENT WORDS AFTER CHECKING FOR  SAME ROOT.                */

                  /* RECORD FORMAT:
                          POSITIONS 01-02  = WORD LENGTH          (LENWD1)
                          POSITIONS 03-07  = MATCH COUNT          (MATCNT)
                          POSITIONS 08-12  = FREQUENCE OF OCCURRANCE  (FREQ)
                          POSITIONS 13-70  = WORD, VARYING IN LENGTH UP TO 58 CHARS    */

                  /* MINIMUM RECORD PHYCICAL LENGTH IN BLOCKED RECORD FORMAT WILL BE
                        21 BYTES:   4 BYTES BLOCK COUNT,
                                    4 BYTES LOGICAL RECORD COUNT
                                   13 BYTES MINIMUM RECORD LENGTH
                                   --
                                   21 BYTES MINIMUM PHYSICAL RECORD LENGTH.

                        MUST USE:  DCB=(RECFM=VB,LRECL=74,BLKSIZE=3556),
                                   DDNAME = 'OUTPUT'

                        BLOCK SIZE OF 3556 IS NOT REQUIRED AS LONG AS IT IS A MULTIPLE OF
                        74 PLUS 4.  TO OBTAIN MINIMUM PHYSICAL RECORD LENGTH OF 21, THE
                        KEYWORD RECFM MUST BE = VB.                                 */


32                      CALL FIRSTWORD;
33                START: SWD = WORKWD;
34                      L = 0;
35                      GO TO OBTAIN2;

36                OBTAIN:
37                      IF EOFSW >= 1 THEN DO;
38                        EOFSW = 2;
39                        GO TO MATCHCK;
40                        END;

41                      CALL CONTENTPROC;

42                OBTAIN2:  IF SUBSTR(WORKWD,1,3) = SWD THEN DO;
44                            L = L + 1;
45                            ROOTCK(L) = CONTENTWD;
46                            GO TO OBTAIN;
47                          END;
```

```
49              MATCHCK:  IF L = 1 THEN DO;
50                          PUT STRING(OUTAREA) EDIT (CNST,'    ',LENWD(1),MATCNT,
                              OCCUR(1),WD(1)) (F(2),A(4),F(2),F(5),F(5),A(LENWD(1)));
51                          CALL PUTOUT (OUTAREA);    /* PUT RECORD TO DATA SET      */
52                          PUT FILE(SYSPRINT) EDIT (MATCNT,WD(1),OCCUR(1))
                              (SKIP,F(8),X(2),A(LENWD(1)),COLUMN(70),F(5));
53                          MATCNT = MATCNT + 1;
54                          IF EOFSW = 2 THEN GO TO EOF;
56                          GO TO START;
57                          END;

                        /* STAPT M AT THE TOP AND GO TO THE BOTTOM */                  0181
                        /* START N AT THE BOTTOM AND GO TO M */                        0182
                        /* LOOKING FOR PAIRS OF WORDS THAT ARE OF THE SAME ROOT */     0183
58                        DO M = 1 TO L-1;
59                          DO N = L TO M + 1 BY -1;
60                            CALL STEM(LENWD(M),WD(M),LENWD(N),WD(N));
61                            IF SAME_ROOT = 1 THEN DO;
63                              IF MAT(M) = 0 THEN DO;
65                                MAT(M) = MATCNT;
66                                MATCNT = MATCNT + 1;
67                                END;
68                              MAT(N) = MAT(M);
69                              END;
70                            END;
71                          IF MAT(M) = 0 THEN DO;
73                            MAT(M) = MATCNT;
74                            MATCNT = MATCNT + 1;
75                            END;
76                          END;
77                        IF MAT(L) = 0 THEN DO;
79                          MAT(L) = MATCNT;
80                          MATCNT = MATCNT + 1;
81                          END;

82                      DO M = 1 TO L;
83                        PUT STRING(OUTAREA) EDIT (CNST,'    ',ROOTCK(M))
                            (F(2),A(4),F(2),F(5),F(5),A(LENWD(M)));
84                        CALL PUTOUT (OUTAREA);      /* PUT RECORD TO DATA SET      */
85                        IF MAT(M) = MAT(M-1) THEN
86                          PUT FILE(SYSPRINT) EDIT (WD(M),OCCUR(M)) (SKIP,COLUMN(11),
                              A(LENWD(M)),COLUMN(70),F(5));
```

```
87                      ELSE
87                        PUT FILE(SYSPRINT) EDIT (MAT(M),WD(M),OCCUR(M)) (SKIP,
                            F(8),X(2),A(LENWD(M)),COLUMN(70),F(5));

88                      END;

99                    IF POPSW = 2 THEN GO TO EOF;
91                    GO TO START;


92          EOF:      CLOSE FILE(SEXT);
93                    CALL CLSEOUT;                 /* CLOSE DATA SET              */
94                    GO TO SUFEND;


                  /*************************************************************************/
                  /*************************************************************************/

                  /* BEGIN SUBROUTINES.                                                    */

                  /*************************************************************************/

95                CONTENTPROC:  PROCEDURE;

                  /*  PROCEDURE COMPARES CONTENT WORDS TO FUNCTION WORDS ELIMINATING
                      CONTENT WORDS WHICH ARE FUNCTION WORDS AND COUNTING THE NUMBER
                      OF OCCURRANCES OF A CONTENT WORD.                                  */

96                        WORKFREQ = 0;  /* SET COUNTER TO 0                           */
97                COMPARE1:
98                        IF FUN(IA) = READWD THEN DO;
99                FIRSTWORD: ENTRY;
100                           ON ENDFILE (SEXT) GO TO RTN;
102                           READ FILE(SEXT) INTO (TEMP1);
103                           GET STRING(TEMP1) EDIT (NUMC,READWD)  (F(2),X(16),A(NUMC));
104                           GO TO COMPARE1;
105                           END;

106                       ELSE DO;
107                           WORKWD = READWD;
108                           WORKLGTH = NUMC;
109                           END;

110                       IF IA <= NFUN THEN DO;
112                           IF FUN(IA) <= READWD THEN DO;
114                               IA = IA + 1;
115                               GO TO COMPARE1;
116                               END;
117                           END;

                  /*        DEFAULT IS FUN(IA) > READWD                                 */
118               COMPARE2:   IF READWD = WORKWD THEN DO;
120                               WORKFREQ = WORKFREQ + 1;
```

```
                                                                    135
121                         ON ENDFILE (SEXT) GO TO RTN;
123                         READ FILE(SFXT) INTO (TEMP1);
124                         GET STRING(TEMP1) EDIT (NUMC,READWD) (F(2),X(16),
                                A(NUMC));
125                         GO TO COMPARE2;
126                         END;

127               RTNA:
                          RETURN;

128               RTN:    EOFSW = 1;
129                       GO TO RTNA;

130                       END CONTENTPROC;

                  /*************************************************************************/

131               STEM:   PROCEDURE (LMA,WDMA,LNA,WDNA);

132                     DECLARE  /* FIELDS USED ONLY WITHIN SUBROUTINE                */
                          LM FIXED DEC (2),          /* LENGTH OF WORD WDM          */
                          LMA FIXED DEC (2),         /* PASSED LENGTH OF WDM        */
                          LN FIXED DEC (2),          /* LENGTH OF WORD WDN          */
                          LNA FIXED DEC (2),         /* PASSED LENGTH OF WDN        */
                          WDM CHAR (58) VARYING,     /* WD(M)                       */
                          WDMA CHAR (58) VARYING,    /* PASSED WD(M)                */
                          WDN CHAR (58) VARYING,     /* WD(N)                       */
                          WDNA CHAR (58) VARYING,    /* PASSED WD(N)                */
                          REMAIN1 CHAR (8) VARYING,  /* POSSILBE SUFFIX 1           */
                          REMAIN2 CHAR (8) VARYING,  /* POSSIBLE SUFFIX 2           */
                          LDIFF BINARY FIXED (15,0), /* LENGTH OF POSSIBLE ROOT     */
                          RDIFF BINARY FIXED (15,0), /* LENGTH OF POSSIBLE ROOT 2   */
                          TEMP CHAR (8);             /* TEMPERORY WORK AREA         */

133                       SAME_ROOT = 0;
134                       LM = LMA;   /* LENGTH MOVED TO NEW FIELD SO THAT ORIGINAL    */
135                       LN = LNA;   /* LENGTH NOT DISTURBED.                         */
136                       WDM = WDMA; /* WORDS MOVED TO NEW FIELD SO THAT ORGINIAL NOT */
137                       WDN = WDNA; /* DISTURBED.                                    */

                  /* CHECK FOR ' , SINGLE 'S', ''S', OR 'SS' ENDINGS.                  */

138                       IF SUBSTR(WDM,LM,1) ='''' THEN DO;
140                         SUBSTR(WDM,LM,1) = ' ';
141                         LM = LM - 1;
142                         END;

143                       IF SUBSTR(WDN,LN,1) ='''' THEN DO;
145                         SUBSTR(WDN,LN,1) = ' ';
146                         LN = LN - 1;
147                         END;

148                       IF SUBSTR(WDM,LM,1) = 'S' THEN DO;
```

```
150                     IF SUBSTR(WDM,LM-1,1) = 'S' THEN GO TO CKWDN;
152                     IF SUBSTR(WDM,LM-1,1) ='''' THEN DO;
154                       SUBSTR(WDM,LM-1,2) = '  ';
155                       LM = LM - 2;
156                       GO TO CKWDN;
157                       END;
158                     SUBSTR(WDM,LM,1) = ' ';
159                     LM = LM - 1;
160                     END;

161             CKWDN:  IF SUBSTR(WDN,LN,1) = 'S' THEN DO;
163                       IF SUBSTR(WDN,LN-1,1) = 'S' THEN GO TO CKSAME;
165                       IF SUBSTR(WDN,LN-1,1)='''' THEN DO;
167                         SUBSTR(WDN,LN-1,2) = '  ';
168                         LN = LN - 2;
169                         GO TO CKSAME;
170                         END;
171                       SUBSTR(WDN,LN,1) = ' ';
172                       LN = LN - 1;
173                       END;

174             CKSAME: IF SUBSTR(WDM,1,LM) = SUBSTR(WDN,1,LN) THEN GO TO SAME;
176                     DO IB = 4 TO LM+1;    /* FIND POINT OF DEVIATION          */
177                       IF SUBSTR(WDM,IB,1) ¬= SUBSTR(WDN,IB,1) THEN DO;
179                         LDIFF = IB;
180                         RDIFF = IB;
181                         GO TO DEVIATE;
182                         END;
183                       END;

184             DEVIATE:
                        IF SUBSTR(WDM,LDIFF,1) = SUBSTR(WDM,LDIFF-1,1) THEN
185                       LDIFF = LDIFF + 1;
186                     IF SUBSTR(WDN,RDIFF,1) = SUBSTR(WDN,RDIFF-1,1) THEN
187                       RDIFF = RDIFF + 1;
188                     IF (LM-LDIFF) > 8 | (LN-RDIFF) > 8 THEN RETURN;
190                     REMAIN1 = SUBSTR(WDM,LDIFF,LM-LDIFF+1);
191                     REMAIN2 = SUBSTR(WDN,RDIFF,LN-RDIFF+1);
192                     IF REMAIN1 > REMAIN2 THEN DO;                          0089
194                        TEMP = REMAIN1;  /* SORT FOR COMPARE AGAINST SUF1 */ 0090
195                        REMAIN1 = REMAIN2;                                  0091
196                        REMAIN2 = TEMP;                                     0092
197                        END;                                                0093

                    /* DO BINARY SEARCH FOR SUFFIX 1 IN SUFFIX1 STRUCTURE ARRAY  */

199                     JC = 0;
199                     KC = NSUF1 + 1;
200             PICK1:  IC = (JC + KC) / 2;
201                     IF REMAIN1 = SUF1(IC) THEN GO TO CKSUF2;
203                     IF IC = JC | IC = KC THEN RETURN;
205                     IF REMAIN1 > SUF1(IC) THEN DO;
207                       JC = IC;
```

```
208                       GO TO PICK1;
209                       END;
210                     KC = IC;
211                     GO TO PICK1;

                /* SUFFIX 1 FOUND, BINARY SEARCH FOR SUFFIX 2 IN SUFFIX2 STRUCTURE   */

212             CKSUF2: JC = LOCSUF2(IC);
213                     KC = LOCSUF2(IC + 1);
214             PICK2:  IC = (JC + KC) / 2;
215                     IF REMAIN2 = SUF2(IC) THEN GO TO CKEXCPT;
217                     IF IC = KC | IC = JC THEN RETURN;
219                     IF REMAIN2 > SUF2(IC) THEN DO;
221                       JC = IC;
222                       GO TO PICK2;
223                       END;
224                     KC = IC;
225                     GO TO PICK2;

                /* SUFFIX 2 FOUND, DO TABLE LOOKUP FOR EXCEPTION WORD                */

226             CKEXCPT: IF LOCEXC(IC) = 0 THEN GO TO SAME;
228                     JC = LOCEXC(IC);
229                     IF LENEXC(JC) = 2 THEN GO TO LETTER_RULE;
231             PICK3:  IF SUBSTR(WORD,1,LM) = EXCPT(JC) THEN RETURN;
233                     IF SUBSTR(WORD,1,LN) = EXCPT(JC) THEN RETURN;
235                     JC = JC + 1;
236                     IF EXCPT(JC) = '0' THEN GO TO SAME;
238                     GO TO PICK3;

239             SAME: SAME_ROOT = 1;
240                     RETURN;

241             LETTER_RULE:      /* CHECK FOR COMMON LETTER                         */
242                     IF SUBSTR(EXCPT(JC),2,1) = SUBSTR(WORD,LDIFF-1,1) THEN GO TO
                                                                          PICK3;
243                     JC = JC + 1;
244                     IF LENEXC(JC) = 2 THEN GO TO LETTER_RULE;
246                     RETURN;

247                   END STEM;

                /*********************************************************************/

                /* END OF SUBROUTINES.                                               */

                /*********************************************************************/
                /*********************************************************************/

248             SUFEND: END SUFFIX;
```

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Department of Information Science University of North Carolina, Chapel Hill, N. C.	Unclassified
	2b GROUP

3. REPORT TITLE

Automated Language Analysis, 1967 - 1968: Report on Research for the period Match 1, 1967 - February 29, 1968.

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (Last name, first name, initial)

Sedelow, Sally Yeates

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
1 March, 1968	137	

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
N00014 - 67 - A - 0321-0-0001 b. PROJECT NO Task No. NR 348-005 c. Office of Naval Research, U. S. Navy d.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

This report describes current research associated with the automated language analysis project. The focus of the research is upon the delineation of patterns formed in the linguistic coding of information; this delineation is called stylistic analysis. The report describes research on thesauri, especially research upon comparisons of possible input thesauri, upon methods of enlarging input thesauri, and upon the possible use of input thesauri for the resolution of semantic ambiguity. The report also describes a new ring-structure version of the VIA program which produces text-specific thesauri. Work on statistical approaches to the analysis of strings is also described. The report contains extensive documentation for the PL/1 implementations of the INDEX, SUFUN, and SUFFIX sections of the VIA program, as well as a description of the program design for the ring-structure version of VIA.

DD FORM 1 JAN 64 1473 0101-807-6800

Unclassified

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Stylistic analysis Automated language analysis Thesaurus Content Analysis						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS), (S), (C), or (U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.