January 1973

Implementation of an Automatic, *a Posteriori*, Hierarchical Classification System

Allen L. Lang
University of Pennsylvania

Susan W. Zagorsky
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-73-05.

Implementation of an Automatic, *a Posteriori*, Hierarchical Classification System

Abstract

This paper describes a total system which provides the capability to semi-automatically index and classify any given file of information. The semi-automatic indexing method assigns key terms to each "document" in the file . These key terms may be modified, (i.e., added, deleted, or changed) by the user. The indexed documents are then assigned to categories by an automatic classification algorithm. The classification assignments are *a posteriori*. Classification dictionaries are also produced which can be used as an aid in browsing through the data base and in retrievals from the data base.

Samples are given of the results obtained while indexing and classifying an experimental data base containing the texts of 1669 radio messages.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-73-05.

University of Pennsylvania
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING

Technical Report

IMPLEMENTATION OF AN AUTOMATIC, A POSTERIORI, HIERARCHICAL CLASSIFICATION SYSTEM

by

Allen L. Lang

and

Susan W. Zagorsky

Project Supervisor
Noah S. Prywes

January 1973

Prepared for the
Office of Naval Research
Information Systems
Arlington, Va. 22217

under

Contract N00014-67-A-0216-0014
Project No. NR 049-153

University of Pennsylvania
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING

Technical Report

IMPLEMENTATION OF AN AUTOMATIC, A POSTERIORI, HIERARCHICAL CLASSIFICATION SYSTEM

by

Allen L. Lang

and

Susan W. Zagorsky

Project Supervisor
Noah S. Prywes

January 1973

Prepared for the
Office of Naval Research
Information Systems
Arlington, Va. 22217

under

Contract N00014-67-A-0216-0014
Project No. NR 049-153

Moore School Report No. 73-05

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
University of Pennsylvania The Moore School of Electrical Engineering Philadelphia, Pa. 19174	UNCLASSIFIED
	2b. GROUP

3. REPORT TITLE

IMPLEMENTATION OF AN AUTOMATIC, A POSTERIORI, HIERARCHICAL CLASSIFICATION SYSTEM

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

Technical Report

5. AUTHOR(S) (First name, middle initial, last name)

Allen L. Lang and
Susan W. Zagorsky

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
January 1973	292	10

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
N00014-67-A-0216-0014	Moore School Report No. 73-05
b. PROJECT NO. NR 049-153	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

Reproduction in whole or in part is permitted for any purpose of the United States Government

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Office of Naval Research Information Systems Arlington, Virginia 22217

13. ABSTRACT

This paper describes a total system which provides the capability to semi-automatically index and classify any given file of information. The semi-automatic indexing method assigns key terms to each "document" in the file. These key terms may be modified, (i.e., added, deleted, or changed) by the user. The indexed documents are then assigned to categories by an automatic classification algorithm. The classification assignments are a posteriori. Classification dictionaries are also produced which can be used as an aid in browsing through the data base and in retrievals from the data base.

Samples are given of the results obtained while indexing and classifying an experimental data base containing the texts of 1669 radio messages.

TABLE OF CONTENTS

FIGURES

TABLES

BIBLIOGRAPHY

1. Borko, H., "The Conceptual Foundations of Information Systems," Systems Development Corporation, Report No. SP-2057:1-37 (May 6, 1965).

2. Lefkovitz, D., File Structures for On-Line Systems (Spartan Books, March 1969).

3. Litofsky, B., "Utility of Automatic Classification Systems for Information Storage and Retrieval," Ph.D. Dissertation, The Moore School of Electrical Engineering, University of Pennsylvania (May 1969).

4. Price, D. J. de S., Science Since Babylon (New Haven, Yale University Press, 1961), page 97.

5. Price, D. J. de S., "Nations can Publish or Perish," Science and Technology (Oct. 1967): 84-99.

6. Prywes, N. S., "Browsing in an Automated Library Through Remote Access," Computer Augmentation of Human Reasoning (June 1964) pp. 105-130.

7. Prywes, N. S., "On-Line Information Storage and Retrieval," AGARD Symp. on Storage and Retrieval of Information (June 18-21, 1968): 1-18.

8. Prywes, N. S., "Structure and Organization of Very Large Data Bases," Proc. Symp. on Critical Factors in Data Management, UCLA (March 1968).

9. Salton, G., The Smart Retrieval System, Experiments in Automatic Document Processing (Prentice-Hall, Inc., 1971), pp. 143-180.

10. Thompson, D. A., Bennigson, L., and Whitman, D., "A Proposed Structure for Displayed Information to Minimize Search Time Through a Data Base," American Documentation (Jan. 1968): 80-84.

CHAPTER 1

1.1 Purpose of This System

It has been estimated that 90% of all journals have been published in the last 50 years; we are in the midst of an "information explosion." There are about 350,000 scientific papers published yearly and we are rapidly approaching 100,000 journals per year [4, 5]

In 1830, when there were only 300 journals published, the solution to keeping up with the increasing number of papers was the publication of the first abstract journal. Now we have reached a point where an abstract of abstracts will not come close to solving the "information explosion" problem.

Some method of collecting, organizing and ultimately selectively retrieving data from any collection of information is needed. Any person involved in work which must utilize the vast amount of literature published on some subject, is faced with the mounting problem of keeping up with the large number of papers and journals published.

This paper presents a system which used by itself can be an aid to any user who has a collection of data and needs some method to organize it so that it can be used more effectively.

Any collection of data, no matter what the size, can be thought of as a "library." The term library, not only refers to collections of journals and books, but also to a variety of other types of information such as individual scientific facts, a collection of law cases, or in the example of Chapters Two and Three, the texts of

foreign news broadcasts. Any "library" can be thought of as consisting of a set of "documents," where a "document" is any unit of information.

Any set of documents can be input to this system, which will semi-automatically extract terms (descriptors) from each "document," classify the documents into categories on the basis of these descriptors, arrange the classified documents into cells, (the contents of each cell containing documents that are most alike) and produce a set of classification dictionaries which aid in browsing through the data base and also aid in retrieving pertinent documents from the data base.

Once the documents have been indexed and classified by this system, the contents of each classification cell (or category) can be placed onto an easily accessible media (i.e. the contents of each cell can be listed on a line printer, stored on computer disk or placed onto microfilm). Once pointed toward a particular page in the listing, area of the disk or section of microfilm, by the classification dictionaries, the user could browse through the documents in that category to find pertinent documents. This browsing and retrieval can be either manual if the storage media is microfilm or printouts, or more importantly it can be automatic if this data is stored on a disk and used in conjunction with an automatic retrieval system.

The experimental data base used in the implementation of this system was put on microfilm along with its associated dictionaries in order to facilitate browsing and retrieval.

The method of this system can be logically extended to the more far reaching goal of completely automating the library as we know it. Prywes points out [7] that the indexing and cataloging functions are the major bottlenecks in most large libraries: "In any one of the large libraries or information centers there are thousands of monographs and serials that are waiting to be catalogued and indexed. These often lay unused because of the dearth of competent catalogues and indexes, especially those expert in particular subjects and languages."

In order to break this bottleneck, the indexing and cataloging of documents must be automated. The computer can be used in processing natural language text for indexing, and automatic classification can be performed for cataloging. The current state of the art in data management and information storage and retrieval by computer, along with the automation of the indexing and cataloging functions, can effectively eliminate the bottleneck and eventually lead to the all-automatic processing of large libraries and data bases. The routines developed in this system could function as one of the many subroutines needed to obtain this goal. Much more research is needed in the area of text processing to develop further the procedure begun here.

1.2 Advantages of Automatic Classification

This paper deals primarily with the implementation of an automatic classification algorithm which can be used to perform the cataloging function in a "library." A document classification algorithm represents a scheme for placing documents on shelves, in microfilms, in bibliographic publications, or in our case, into the computer. The goal of any document classification is to group "like" documents

together into categories, where "likeness" is defined by the respective classification algorithm used.

Before a document collection can be classified, however, it must be indexed. The purpose of the indexing function is to obtain a number of descriptors which can act as a surrogate for each document. As was mentioned in the previous section, it is the classification and indexing functions that are the major bottlenecks in most large libraries and must be automated in order to realize all-automatic library processing.

In a conventional library, documents on a common subject are grouped on the same or adjacent shelves. On the other hand, an automatic classification algorithm will place the document surrogates created by the indexing function into convenient units of the computer's memory such as disk cylinders or magnetic strips. Each of these units of memory will contain only "like" documents and will be called cells.

In the realm of automatic classification one can identify two levels of automation. The first level is the placement of documents into a priori categories. That is, the categories and sub-categories to contain "like" documents are decided upon before the documents are actually classified.

The other level of classification is the use of automated techniques to derive the classification categories a posteriori. In other words, documents are first placed into cells, grouping "like" documents together in the same cell. ("Likeness" may be defined by the number of descriptors or keywords common to two document surrogates.) After every document has been assigned to a cell, the classification

categories are defined precisely by the contents of each cell. One can see that an a priori classification requires the documents to be partitioned on the basis of some pre-defined or "natural" divisions of knowledge; whereas, the a posteriori technique actually optimizes the classification categories with respect to the documents existing in the collection.

Litofsky [3] compares both levels of classification techniques and lists the following major advantages of an automatic, a posteriori, heirarchical classification:

1. Directory Size Reduction. The inverted file directory can be reduced by more than an order of magnitude. This can be accomplished by forming an inverted file directory on the classification cells, rather than on the individual documents.

2. Reduction in Memory Accesses. "Like" documents are grouped into cells which are segments of mass storage (tracks, cylinders, magnetic strips) that do not require more than one physical access motion. Since the transmission time for an entire cell is usually much smaller than the average access motion time, it costs little extra in time to retrieve all surrogates within a cell than it would to retrieve a single surrogate. The document surrogates in a given cell are, by definition, "alike"; therefore, there is a high probability that multiple retrievals for a given query would appear in the same cell. This reduces the number of cells accessed per query and hence the total number of memory accesses required.

3. Flexibility. With an a posteriori classification, the categories are decided upon after all documents have been classified. The resulting classification is therefore specifically tailored to the individual user's document collection rather than requiring the document to fit into a priori categories. Coupled with the automatic nature of the classification process, this leads to a large degree of flexibility and ability to maintain up-to-date classification schedules.

4. Browsability. The ability to browse through parts of a collection should be an essential portion of any library, especially an all-automatic library. In "The Conceptual Foundations of Information Systems," Borko [1] notes: "The user searches for items that are interesting, original, or stimulating. No one can find these for him; he must be able to

> browse through the data himself. In a library, he wanders among the shelves picking up documents that strike his fancy. An automated information system must provide similar capabilities."
>
> Effective browsing demands a heirarchical classification system in order to enable one to start with broad categories and work towards specific. In a posteriori classification system the hierarchy is formed by grouping "like" documents into cells, "like" cells into groups of cells, etc., until all documents are in one large group: the entire collection itself. The hierarchy of descriptors is formed from the bottom (cell) to the top (entire collection) of the hierarchy. The node names in each level of the hierarchy are generated automatically and consist of the set of descriptors which appear in all of the nodes directly beneath the node in question. The resulting set of descriptors can be considered an "abstract" [6] of the knowledge contained beneath that node in the tree (thinking of the hierarchy as an inverted tree). Classification schedules are required in order to be able to make use of a hierarchically classified document collection. These schedules consist of what shall be called a "node-to-key" table and a "key-to-node" table. The node-to-key table is analogous to the Dewey decimal classification schedule where "node" 621.3 points to the "key" Electrical Engineering. The key-to-node table performs the inverse function, that of producing node numbers corresponding to given keywords or descriptors.

1.3 Implementation of an Automatic Classification Algorithm

This paper describes the implementation of an automatic classification algorithm. This algorithm, which was conceived by Lefkovitz [2] is of the a posteriori type and produces a hierarchical classification suitable for efficient browsing.

The classification algorithm is first applied to the entire document collection (i.e., the top level of the hierarchy) and results in the partitioning of the collection into groups of "like" documents. These groups, each of which will contain many documents, constitute the next level of the hierarchy. In order to further develop the hierarchy, the algorithm is then re-applied to each one of the groups in turn. The

process will terminate when all groups meet a cell size criteria. One can see that this partitioning and re-partitioning of the collection will produce a tree structure with the entire collection at the top level and cells that meet the group size criteria at the bottom.

The classification algorithm requires that a surrogate be created for each document. These surrogates must contain the descriptors assigned to each document by an indexing function. In order to transform each source document into a surrogate, a "semi"-automatic indexing algorithm is also implemented.

The indexing algorithm is "semi"-automatic in that it does not make all of the decisions necessary to create a set of descriptors, and thus a surrogate, for each source document. The user must interface with the indexing routines and has final judgement as to the contents of each surrogate. In this way, the semi-automatic indexing routines function as a tool, aiding the user in the assignment of descriptors to each document. While the indexing algorithm described in this paper will produce a set of descriptors for each document, the reader should not confuse this algorithm with the automatic indexing function required for an all-automatic library. The indexing algorithm of this paper is intended only as a preprocessor for the automatic classification routines and is by no means fully automatic. An automatic indexing algorithm will create descriptors for each document without any intervention by a user; the indexing algorithm of this paper <u>requires</u> user intervention. The user has routines available to him for changing, adding or deleting descriptors of a document.

1.4 Note on the Computer Programs

The automatic classification and semi-automatic indexing algorithms described in this paper are implemented in FORTRAN; therefore, they are somewhat independent of the particular computer being used. The following is a list of suggestions that a user of these computer programs must be aware of:

1. The user's computer must have at least four tape or one disk drive and 132k bytes of virtual memory (1 byte = 8 bits).

2. A sort package must be provided in order to sort several of the intermediate files created.

3. The FORTRAN unit numbers of 5 and 6 are used as the card input device and line printer respectively.

4. In order to run any of the programs, all the user need do is define his input and output files and supply any required input cards. Complete program and file descriptions are given in the next two chapters.

The semi-automatic indexing routines are described in Chapter Two. These programs accept the user's source documents as input, assign a set of descriptors to each document, and create document surrogates. Also output from the semi-automatic indexing programs is a listing of the unique words in the document collection.

Chapter Four describes the automatic classification routines. These programs take the file of document surrogates created by the semi-automatic indexing routines and classifies them according to an a posteriori, hierarchical algorithm also described in the chapter. Output from the classification is a file containing the document surrogates grouped into the a posteriori categories generated by the classification algorithm. The following two classification schedules

are also output from the automatic classification routines:

1. Node-to-Key Table. This listing displays the keywords (descriptors) assigned to each node in the classification hierarchy. (A node will be assigned many descriptors.)

2. Key-to-Node Table. This listing displays the node numbers corresponding to each unique descriptor. (A descriptor may appear at several different nodes.)

CHAPTER 2

SEMI-AUTOMATIC INDEXING

2.1 Purpose of Semi-Automatic Indexing Routines

The purpose of the semi-automatic indexing routines is to transform the original collection of documents into a collection of document "surrogates" which will be input to the automatic classification routines.

A document surrogate consists of a document number (used to identify the document) and a set of integer codes corresponding to the descriptors (keywords) assigned to each document. If the original document collection has already been indexed, each unique keyword will be assigned an integer code by the semi-automatic indexing routines. If the original document collection is not indexed, the semi-automatic indexing routines will extract pertinent words from each document's text (and/or abstract, title) and assign them to the document as its descriptors. In either case, the keywords are finally replaced by unique code numbers and each document is assigned a document number. This set of codes (document number and keyword codes) constitutes a document surrogate.

2.2 Overview of the Semi-Automatic Indexing Routines

The following eight steps are required to assign a document surrogate to each document (each step will be covered in greater detail in the following sections):

(1) Preparation of the documents for input and assigning each document a number.

(2) Extraction of pertinent words from each document in the collec-

tion.

(3) Sorting (alphabetizing) of words within each document.

(4) Elimination of duplicate words within each document.

(5) Sorting (alphabetizing) of entire collection of extracted words.

(6) Elimination of duplicate words within the entire document collection and assigning each unique word a code number.

(7) Modifying the set of unique words, i.e., making additions, deletions, and changes.

(8) Creation of the surrogate for each document, i.e., replacing the unique words extracted from each document with the corresponding code number.

Step (7) requires the user to manually examine the unique words in order to determine what modifications, if any, must be made. This manual examination is why the indexing process is termed "semi"-automatic.

2.3 Preparation of the Original Document Collection for Input

The original document collection must be placed on a storage medium (magnetic tape, disk, etc.) in a format acceptable to the semi-automatic indexing routines. This format will be referred to as "Standard Input Records" and the storage medium will be called the "Standard Input File." Since all document collections are somewhat unique, it is the user's responsibility to write the computer program required to transform his document collection into Standard Input Records and place these records onto the Standard Input File. This section, which assumes some knowledge of computer programming and file structures on the part of the reader, will describe the Standard Input File, the Standard Input Records on this file, give necessary programming considerations, and

present a general flowchart of the required user written program.

2.3.1 Description of the Standard Input File

The contents of the Standard Input File are used to create the surrogate for each document; therefore, the user must take care as to what information he places on this file. The documents in the user's original collection may consist of title, abstract, full text, keywords, or any combination of these. If the document collection has already been indexed (i.e., there exist keywords for each document in the collection), then the user should be sure to place each document's keywords on the Standard Input File. The user may choose to include more information, for each document, than just its keywords; but if keywords exist, they should be used. If the document collection has not been indexed, then the document's full text, abstract, title, or any combination of these must be placed on the Standard Input File, and the semi-automatic indexing routines will extract pertinent words from the information given and assign them to the document as keywords.

2.3.2 Description of Standard Input Records

Once the user has determined what information is to be placed on the Standard Input File, he must write a computer program to read the given information for each document, block it into the Standard Input Records, and write these records to the Standard Input File.

Each Standard Input Record is a collection of the following four fields (groups) of information:

(1) Information used by the computer's operating system.

(2) Length of the fourth field.

(3) Document number.

(4) Text - this is the information that the user had decided to place on the Standard Input File.

The maximum length of a Standard Input Record is 4096 bytes (1 byte = 1 character = 8 bits). If the information for any document cannot fit into one Standard Input Record, it may span as many records as required, as long as the document numbers in each record spanned by a document are the same. Figure 2.0 gives a complete description of a Standard Input Record.

2.3.3 Programming Considerations

Figure 2.1 shows a general flowchart of the user written program that creates the Standard Input File. Input to this program is the information that the user wishes to associate with each document (full text, title, abstract, keywords, or some combination of these). This information is blocked into Standard Input Record(s), and written to the Standard Input File.

The first field in every Standard Input Record contains information used by the computer's operating system. (This is usually a length of block and length of record value.) The user is not responsible for generating this information in his program; it is usually prefixed to each user generated record by the operating system before the record is output. The user must be aware of the length of this operating system generated field (the length will be referred to as 'S' bytes) in order to insure that the total length of each Standard Input Record is not greater than 4096 bytes.

FIELD #	LENGTH (in bytes)	PROGRAM VARIABLE	FORMAT	DESCRIPTION
1	'S'	——	——	The user is not responsible for creating or reading this field. It is prefixed to every user generated record by the operating system. Note: if this field is not prefixed by O.S. then S=0 in the length computation for field #4.
2	5	LEN	I5 - Zoned Decimal	The length of the fourth field.
3	6	IDNUM	I6 - Zoned Decimal	A sequentially generated number used to identify each document.
4	Not more than: 4085-'S'	ITEXT	EBCDIC characters ('A' format)	Any information that the user wishes to associate with the document may be placed into this field, which is refered to as the 'TEXT' field. (The user may place the documents full text, title, abstract, keywords, or any combination of these items into this field.)

Length of Record: ≤ 4096 Bytes

FIGURE 2.0

DESCRIPTION OF A STANDARD INPUT RECORD

```
WRITE(ITSIF,1ØØ)LEN,IDNUM,(ITEXT(J),J=1,LEN)
1ØØ FORMAT(I5,I6,2Ø(2Ø5A1))
```

Where ITSIF is the Standard Input File number.

field #	variable	description
2	LEN	the length of the fourth field
3	IDNUM	the document number
4	ITEXT	an array containing the information associated with the document and has a maximum dimension of 4085. (Each position of this array will contain one character only.)

As illustrated in the flowchart in Figure 2.1, the user's program generates the last three fields of the Standard Input Record and the computer's operating system will prefix this information with the first field before the record is written to the Standard Input File.

The following example will illustrate the required processing, within the user's program, to generate Standard Input Record(s) for a given document.

Assumptions:

(1) The user has a magnetic tape containing the text of each document in his collection.

(2) The 405th document is being processed and its length is 10,000 bytes (characters).

(3) The length of the field that the operating system prefixes to every record, field number one of the Standard Input Record, is 8 bytes (i.e., S=8).

(4) The maximum length of the fourth field, text, of every Standard Input Record is computed from the formula (4085-S) and is equal to 4077

bytes.

The user's program will extract the first two 4077 byte segments from the 405th document and generate two Standard Input Records of total length 4096 bytes (length of first field = 8 bytes, second field = 5 bytes, third field = 6 bytes, and fourth field = 4077 bytes). The remaining 1846 bytes of the 405th document will be placed into a third Standard Input Record whose total length is 1865 bytes (length of first field = 8 bytes, second = 5 bytes, third = 6 bytes, and fourth = 1846 bytes). Figure 2.2 shows the character and hexidecimal representations of the second and third fields in each of the three generated Standard Input Records. As can be seen from the hexidecimal representations, the user's program generates 5 and 6 byte values for length and document number fields.

2.4 Extraction of Pertinent Words from Each Document in the Collection

Once the user has placed his document collection on the Standard Input File, each document must be analyzed in order to obtain pertinent words which can be assigned to the document as keywords. Program EXTWRD performs this task by scanning the records on the Standard Input File, extracting "words" from the fourth field (Text field) of each Standard Input Record, and saving the extracted "word" on an output file if it is determined that the "word" is pertinent.

2.4.1 Input

Input to program EXTWRD is the Standard Input File and a set of parameter cards. The user, through these parameter cards, defines the

Record	Representation	Field #2 Length = 5 Bytes	Field #3 Length = 6 Bytes
1st	Character	4077	405
	Hexidecimal	40F4F0F7F7	4040404F4F0F5
2nd	Character	4077	405
	Hexidecimal	40F4F0F7F7	4040404F4F0F5
3rd	Character	1846	405
	Hexidecimal	40F1F8F4F6	4040404F4F0F5

Figure 2.2

The contents of the second and third fields in the Standard Input Records generated for the 405th document.

maximum length of an extracted word (any word larger than the maximum will be truncated), a stop list (common words which, if found within a document, are ignored), and a set of word-delimiting characters (characters which signal the end of a word such as the 'blank' character). Figure 2.3 gives a complete description of the required parameter cards.

2.4.2 Output

Output from program EXTWRD consists of a listing of all parameter cards read in, two statistics generated while pertinent words are being extracted, and an output file containing the pertinent words extracted from each document. The two statistics generated are the number of records (one extracted pertinent word per record) on the output file and the number of extracted words eliminated because they were found to be on the user's stop list. These two statistics can be used to rate the user's stop list since, if a document's full text is being scanned, about 1/4 of the total number of words extracted should be found on the stop list and ignored. Each record on the output file, which will be referred to as the 'WORD/DOCUMENT-NUMBER file', consists of a word and the corresponding document number of the Standard Input Record that contained the word. Figure 2.4 gives a complete description of a record on the WORD/DOCUMENT-NUMBER file.

2.4.3 Program Description

Program EXTWRD begins by first extracting "words" from each document. The definition of an extracted "word" is a string of one or more (non-word delimiting) characters from the Text field of a Standard

Card #	Columns	Program Variable	Format	Description
1	1-5	MAXWL	I5	Maximum number of characters per word. (Right justified in the field.) The recommended value is '20', but any value less than or equal to 20 can be used. This value must be consistant throughout all of the Semi-Automatic Indexing routines.
	6	IPRM	I1	This field is used to omit the reading and comparing of extracted words to a stop list. The value of this field must either be $\emptyset$ or 1. If it is 1, then no stoplist is read. If the value of this field is $\emptyset$, then the next card is assumed to contain the length of the stoplist, and the following cards the stoplist itself.
2*	1-5	LNSTL	I5	The number of words on the user defined stoplist. (Right justified in the field.)
(The next 'LNSTL' cards contain the stoplist, one word per card.)				
3*	1-'MAXWL'	ISTOP	EBCDIC Characters	A word on the user defined stoplist left justified in the field (i.e., starting in column one).
4* :	1-'MAXWL'	ISTOP	EBCDIC characters	A word on the user defined stoplist, left justified in the field.
(LNSTL+2)*	1-'MAXWL'	ISTOP	EBCDIC characters	Last word on the user defined stoplist, left justified in the field.

Figure 2.3 Description of Parameter Cards Needed for Program EXTWRD

Card #	Columns	Program Variable	Format	Description
LNSTL+3	1-5	LNDEL	I5	Number of characters on word delimiter list. Right justified in the field.
LNSTL+4	1-'LNDEL'	IDEL	A1	The word delimiting characters. These should include characters such as " ') (> < and the 'blank', one character per column, starting with column one of this card.

* These cards are omitted only if the user punches a '1' in column six (second field) of the first parameter card.

Figure 2.3, continued

Description of Parameter Cards Needed for Program EXTWRD

Field #	Length(in Bytes)	Program Variable	Format	Description
1	2*MAXWL	IWORD	EBCDIC characters	This is the normalized extracted pertinent word. The length of this field is twice the maximum number of characters per word (MAXWL), which is read from the first parameter card. (The characters of each word are stored one character per two bytes, hence length=2*'MAXWL').
2	4	NDN	Fixed Point Binary	This is the document number associated with the extracted word.

Length of Record: (2*MAXWL+4) bytes

Figure 2.4

Description of a Record on the WORD/DOCUMENT-NUMBER File

Input Record that either fall between two word-delimiting characters or between the beginning of the Text field and a word-delimiting character. In order to avoid the premature extraction of the first part of a word that has been split between successive records on the Standard Input File, the string of characters between a delimiting character and the end of the Text field is only considered to be a "word" if the document number in the next Standard Input Record is different from the current record's document number. The user defines his own list of word-delimiting characters through the input parameter cards.

After each "word" has been extracted, program EXTWRD performs several tests to determine if the extracted "word" is pertinent. If the "word" cannot pass all of the tests, it is not pertinent and is ignored; otherwise, it is written to the output file along with its corresponding document number.

The first test, in deciding whether or not an extracted "word" is pertinent, is to examine its length. The extracted "word" is ignored if its length is less than three characters. If the word passes the length test, its last character is examined. If this character is determined to be a "special character," then the character is dropped from the word and the next one is examined. (A "special character" is defined to be any character other than the 26 alphabetics, 10 numerics, and the 'blank' character.) When all trailing "special characters" have been truncated the length test is again performed. The word is ignored if the truncation of trailing "special characters" has reduced its length to fewer than three characters. The test for trailing "special characters" is necessary since any "special character" not on

the user's word-delimiter list could appear as a trailing character and should be truncated. For example, if the period (.) is not defined to be a word-delimiter, then the last "word" extracted from every declarative sentence will have a period as its last character. Since some "special characters" can logically appear in the middle of a word, the user is warned against placing all "special characters" on his word-delimiter list (which would avoid having trailing "special characters" in extracted "words"). By doing so he would cause strings of characters, that would normally be extracted as a "word," to be split and/or deleted. For example, by including the colon (:), hyphen (-), and slash (/) as word-delimiters, the date '$\emptyset$8/$\emptyset$1/72', the time '9:15', and the word 'co-operative' would be split and extracted as the following seven "words": '$\emptyset$8', '$\emptyset$1', '72', '9', '15', 'co', and 'operative'. Of those seven "words," only 'operative' would pass the length test. The other six "words" would be eliminated, thus losing valid information from the document. It is suggested that in order to avoid losing information in this manner, the user include on his word delimiter list only those characters that usually surround and do not appear within a word, e.g., " ') (< > and the 'blank' character.

After the length and trailing "special characger" tests, the extracted word is compared to the user defined stop list. This list contains words that cannot be used as keywords since they have a very high frequency of occurrence and would add little, if any, information to a document. (An example of a stop list is given in Section 2.11, Step 1.) If the extracted word is found to be on the stop list it is ignored; otherwise, the extracted word is considered to be pertinent. The stop list comparison test is optional and is controlled by the

user through the first input parameter card (field #2). If the user's original document collection is indexed, he may choose to place only the document's index terms on the Standard Input File. In this case, there is no need to perform the stop list test since words extracted from the Text field of each Standard Input Record will only be valid index terms and are all pertinent by definition. By punching a '1' in the sixth column of the first parameter card (see Figure 2.3), the user will cause program EXTWRD to bypass the stop list test. It should be noted that if the user decides to bypass the stop list test, he should not include a stop list in the input parameter cards since program EXTWRD will also bypass reading a stop list.

In order to reduce the total number of unique words extracted, each word that passes all previously described tests (i.e., has been determined to be pertinent) is first normalized before it is written to the output file. The normalizing routine, which is a modified version of one used by Litofsky (2), removes a number of different suffixes. A flowchart of this program is given in Figure 2.5. Suffixes deleted are: s, es, ed, ing, ings, ion, ions, ly, edly, ingly, plus a doubled letter immediately follwed by ed or ing. In addition, ies, ied and ily are replaced by the single letter y. It should be noted that the above list merely indicates the suffixes that may be removed under appropriate conditions. The user should consult the flow chart (Figure 2.5) to determine the exact context in which a suffix will be deleted. The normalizing routine will never reduce a word's length below three characters.

After the extracted pertinent word has been normalized, it is

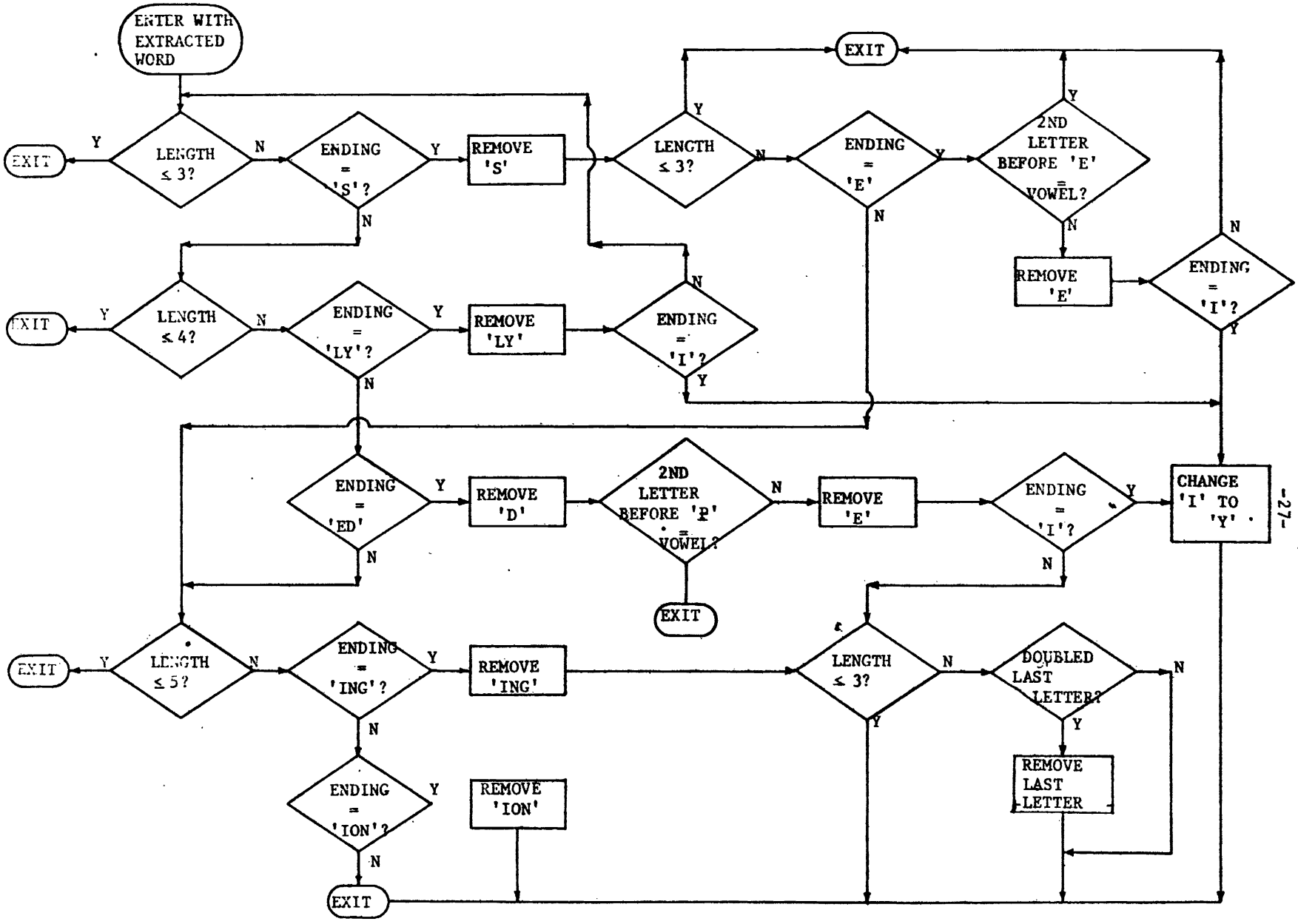


FIGURE 2.5
SUFFIX DELETION ROUTINE

written to the WORD/DOCUMENT-NUMBER file along with its document number (i.e., the document number from the Standard Input Record that contained the word). Figure 2.6 presents a general flow chart of program EXTWRD.

2.5 Sorting (Alphabetizing) of Words Within Each Document

At this point, the words in each document must be sorted in order to eliminate multiple occurrences of words within any document. (A sort routine is not included in the Semi-Automatic Indexing Routines since it is standard at most computer installations.) The WORD/DOCUMENT-NUMBER file, which was output from program EXTWRD, should be input to the user-provided sort routine. The document number must be the major sort field in each record; the characters of each word should be the minor fields. Figure 2.7 summarizes the sort field parameters that must also be input to the sort routine. Output from this sort will be the SORTED-DOCUMENT-NUMBER/WORD file whose format must be identical to the WORD/DOCUMENT-NUMBER file (see Figure 2.4).

2.6 Elimination of Duplicate Words Within Each Document

Once the words within each document have been sorted, multiple occurrences of any word in a document are eliminated by program ELDID.

2.6.1 Input

Input to program ELDID is either the SORTED-DOCUMENT-NUMBER/WORD file or the SORTED-DOCUMENT-NUMBER/WORD/FREQUENCY file (this file will be described in a later section) and a parameter card.

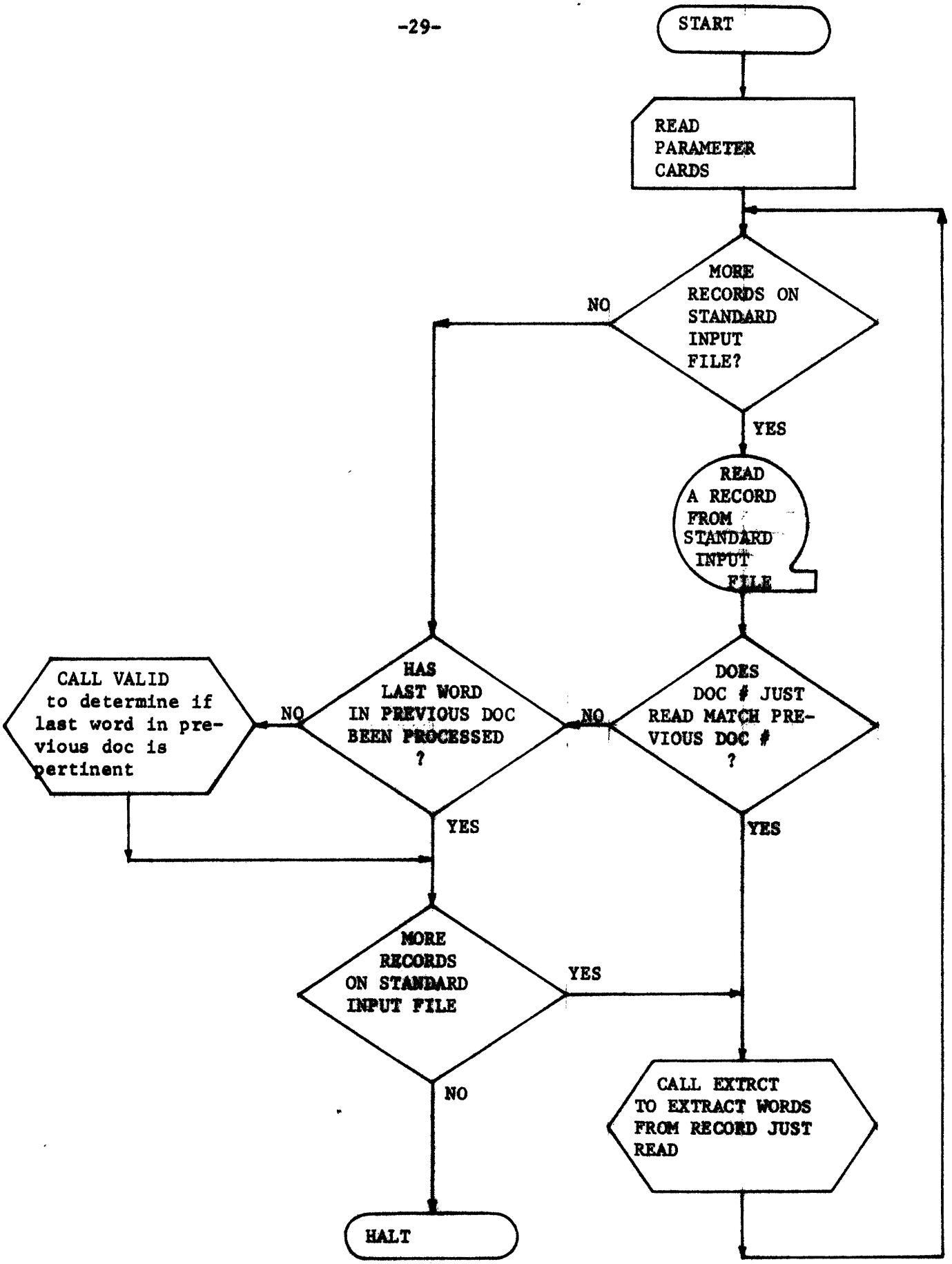


FIGURE 2.6
GENERAL FLOWCHART OF PROGRAM EXTWRD

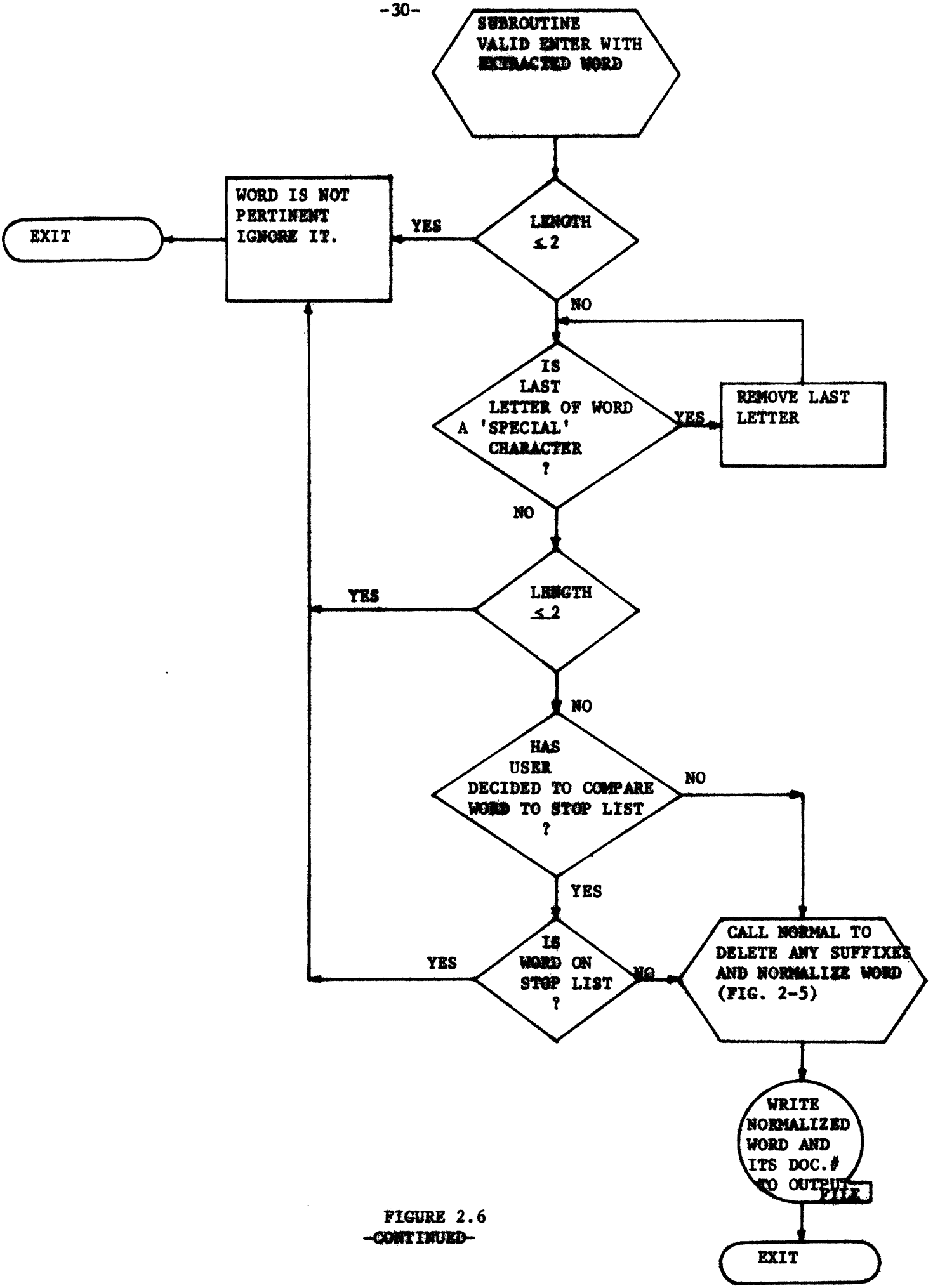


FIGURE 2.6
-CONTINUED-

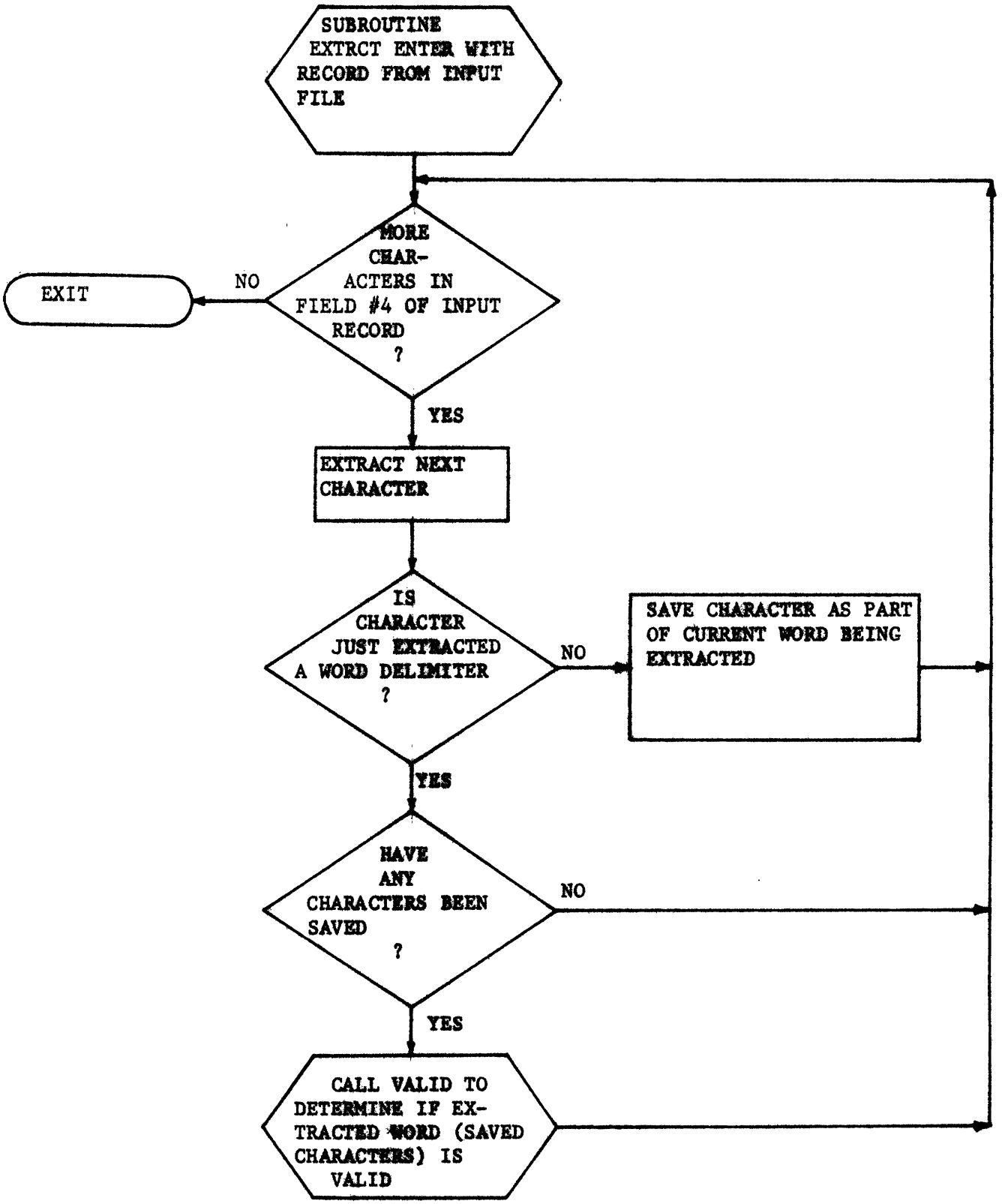


FIGURE 2.6
-CONTINUED-

Field	Type*	Length(Bytes)	Description
1	FI,A	4	Major Field - Document Number
2	FI,A	2	first character of word
3	FI,A	2	second character of word
.	.	.	.
.	.	.	.
.	.	.	.
(MAXWL)**	FI,A	2	next to last character of word
(MAXWL)+1	FI,A	2	last character of word

*FI denotes fixed point integer; A denotes ascending sequence

**MAXWL denotes the maximum number of characters per word

This was an input parameter to program EXTWRD (see Figure 2.3).

Figure 2.7

Description of Sort Field Parameters
Required to Sort Words Within Each Document

The parameter card contains three values: (1) the maximum number of words allowed per document, (2) the maximum number of characters per word, and (3) a parameter to determine which of the two possible input files will be used. Figure 2.8 gives a complete description of the parameter card and Figure 2.9 describes the two possible input files.

The maximum number of words allowed per document may be any value from 1-250, but it is our suggestion that a value of 100 or less be used. For any document with more words than this maximum value, its document number and actual number of words will be printed.

2.6.2 Output

Output from program ELDID is the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file. Each record on this file contains a unique word within a given document, the document's number, and the word's frequency within the document. Figure 2.10 gives a description of this file. Also output is the list of documents with more words than the user defined maximum. Before the Classification Routines can be run, the number of words in these documents must be reduced to below this maximum. (Several utility programs will be described in later sections that will aid the user in reducing the number of words in these documents.) The total number of documents in the collection, and the number of records on the output file are also printed.

2.6.3 Program Description

Program ELDID performs the task of eliminating duplicate words and producing a frequency distribution of the words in each document.

Column	Program Variable	Format	Description
1-3	MAXKS	I3	Maximum number of words allowed per document. For each document with more than the maximum number of words, its document number and actual number of words are printed. The value of this parameter may be from 1-250 and must be right justified in the field. The recommended value is $\leq$ 100.
4-5	MAXWL	I2	Maximum number of characters per word, right justified in the field. This number must be identical to the first field of the first parameter card for program EXTWRD (see Figure 2.3).
6	IPRM	I1	The value of this parameter must either be a $\emptyset$ or 1. If it is a $\emptyset$: the input file is assumed to be the SORTED-DOCUMENT-NUMBER/WORD FREQUENCY file. If it is a 1: the input file is assumed to be the SORTED-DOCUMENT-NUMBER/WORD file (see Figure 2.9 for a description of these two files).

Figure 2.8

Description of Parameter Card for Program ELDID

Value of Parameter IPRM	File Name	Field #	Length (Bytes)	Program Variable	Format	Description
∅	SORTED-DOCUMENT-NUMBER/ WORD/FREQUENCY	1	2*MAXWL	IWORD	EBCDIC characters	Word from a document
		2	4	NDN	Fixed Point Binary	Document number
		3	2	NFQ	Fixed Point Binary	Number of occurrences of the word within the document
	Record Length=2*MAXWL+6					
1	SORTED-DOCUMENT-NUMBER/ WORD	1	2*MAXWL	IWORD	EBCDIC characters	Word from a document
		2	4	NDN	Fixed Point Binary	Document Number
	Record Length=2*MAXWL+4					

Figure 2.9

Description of the Two Possible Input Files to Program ELDID

Field #	Length(Bytes)	Program Variable	Format	Description
1	2*MAXWL	IPWORD	EBCDIC characters	unique word in a document
2	4	IPREVD	Fixed Point Binary	document number
3	2	IWFQ	Fixed Point Binary	Frequency of occurrence of the unique word within the document

Record Length = 2*MAXWL+6

Figure 2.10

Description of UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY File

This program begins by reading a word from the input file. If this current word matches the previous word in the document, a frequency counter is incremented and another word is read. If the current word does not match the previous word in the document, the previous word, its document number, and frequency are written to the output file. The number of words in each document is accumulated. If this accumulated total is greater than the user-defined maximum, then the document's number and the actual number of words in the document are printed. Figure 2.11 presents a general flowchart of program ELDID.

2.7 Sorting of Entire Collection of Extracted Words

Once the duplicate words within each document have been eliminated, the entire collection of words must be sorted. The UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file contains unique words within each document, but a word may appear in several documents; hence this file should be input to the sort routine. The word in each record must be the major sort field; the document number should be the minor field. Output from the sort will be the SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file whose format must be identical to the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file (see Figure 2.10). As noted in Section 2.5, the user is responsible for providing a sort routine. Figure 2.12 summarizes the sort field parameters required by the sort routine in order to sort the entire collection of words.

2.8 Elimination of Duplicate Words Within the Entire Document Collection and Assigning Each Unique Word a Code Number

After sorting the words on the UNIQUE-WORD/

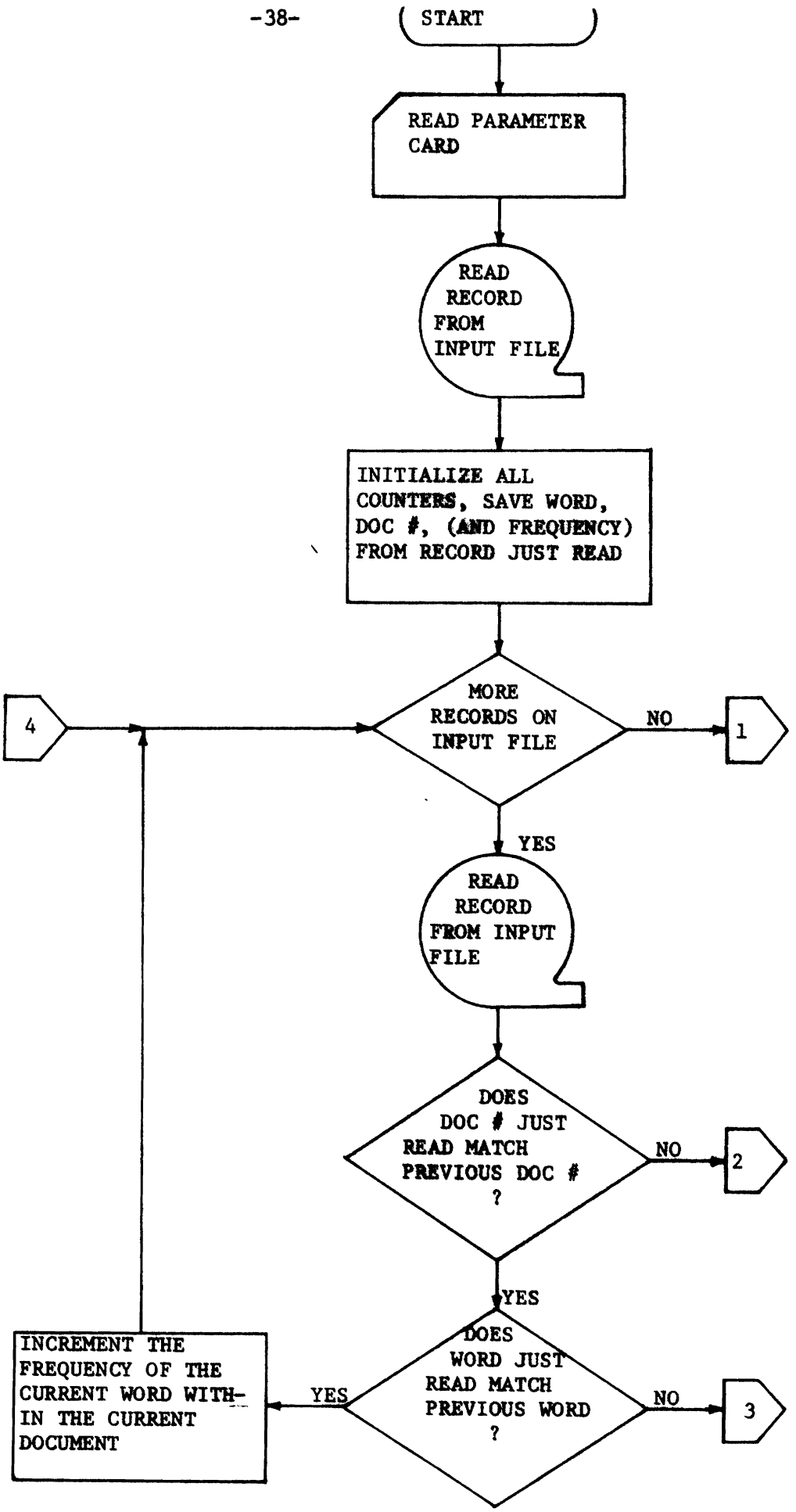


FIGURE 2.11
GENERAL FLOWCHART OF PROGRAM ELDID

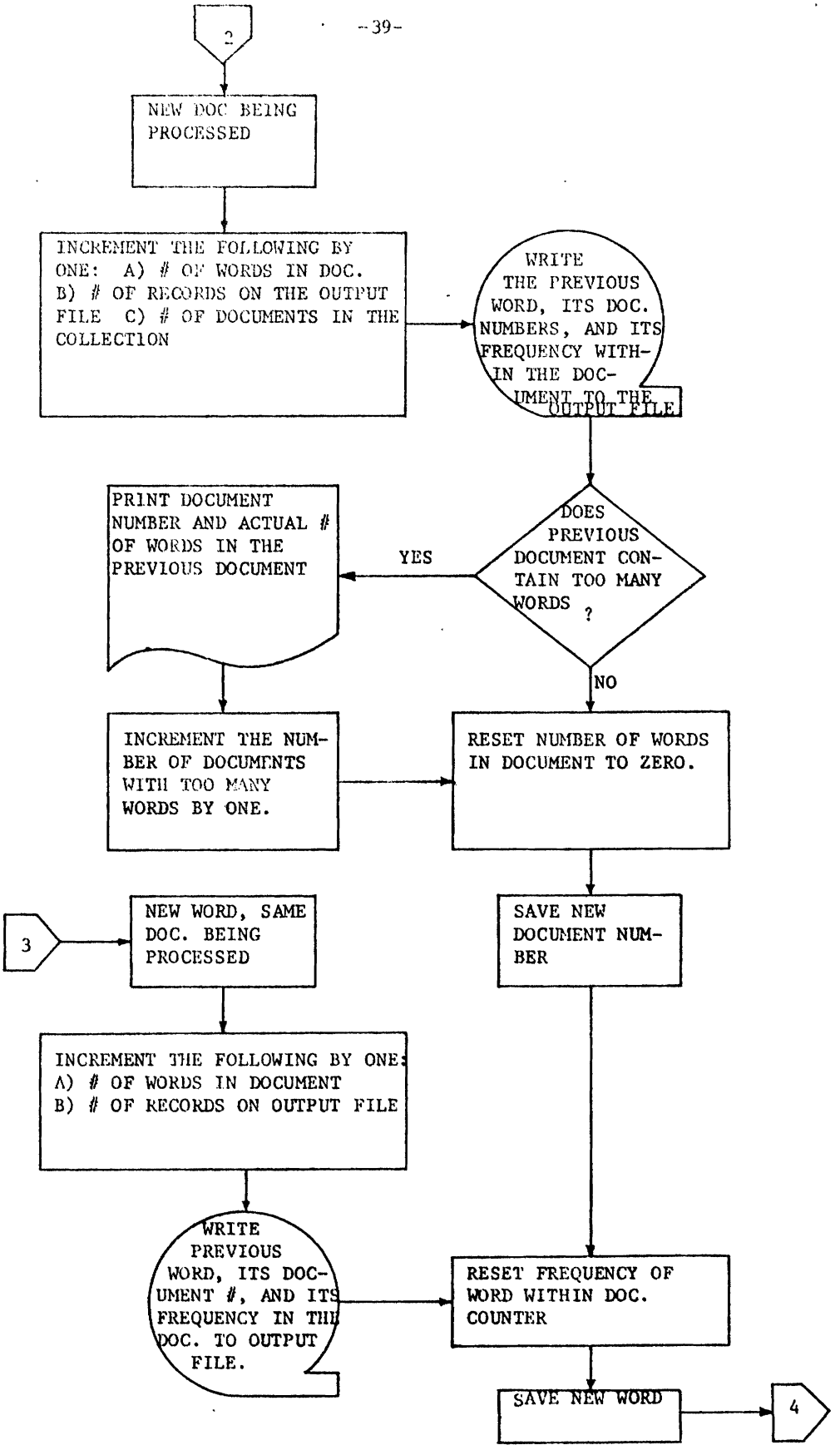


FIGURE 2.11
-CONTINUED-

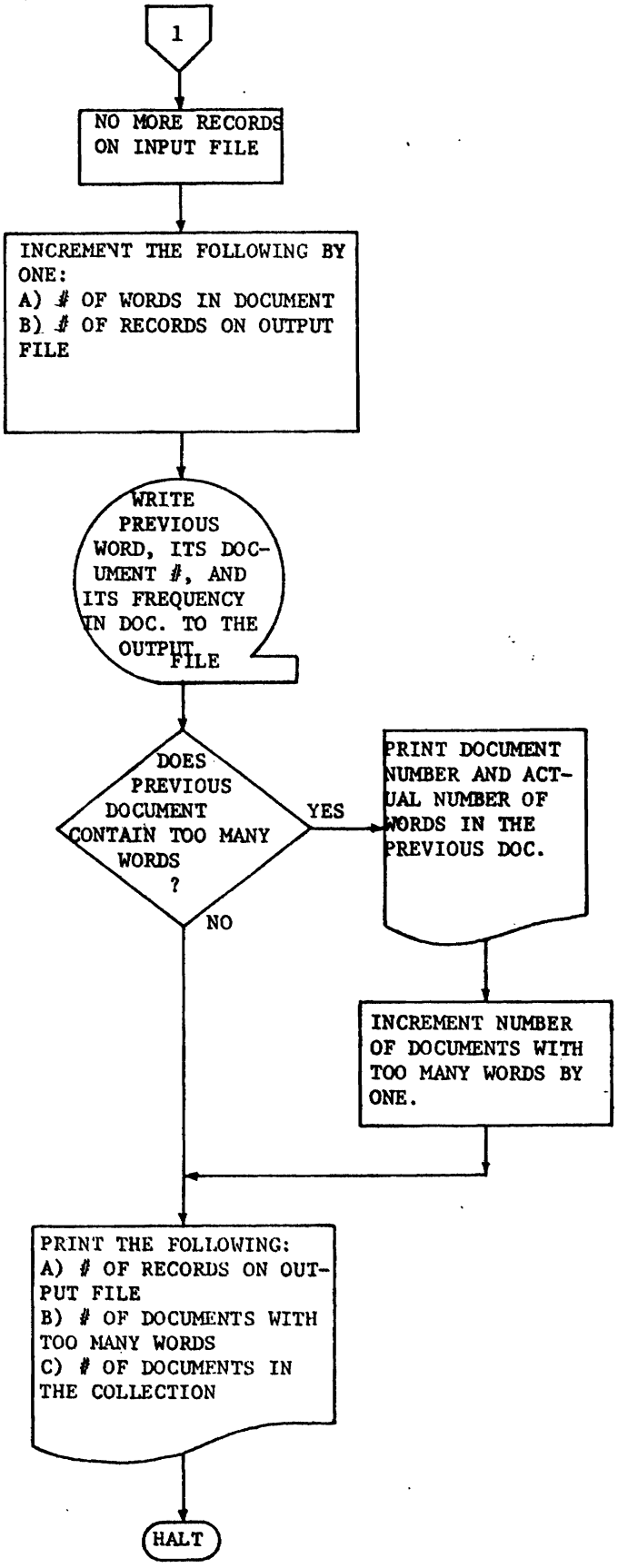


FIGURE 2.11
-CONTINUED-

Field Number	Type*	Length (Bytes)	Description
1	FI,A	2	first character of word
2	FI,A	2	second character of word
3	FI,A	2	third character of word
.	.	.	
.	.	.	
.	.	.	
(MAXWL)**	FI,A	2	last character of word
(MAXWL)+1	FI,A	4	document number

*FI denotes fixed point integer; A denotes ascending sequence

**MAXWL denotes the maximum number of characters per word. This was an input parameter to program ELDID (see Figure 2.8)

Figure 2.12

Description of Sort Field Parameters
Required to Sort the Entire Collection of Words

DOCUMENT-NUMBER/FREQUENCY file, duplicate words within the entire document collection are eliminated and each is assigned a code number. Program UNWRDS performs this task and produces a file containing only unique words.

2.8.1 Input

Input to program UNWRDS is the SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file and a parameter card. The parameter card contains two fields: (1) A parameter to determine which output option to use (see Section 2.8.2), and (2) the maximum number of characters per word. Figure 2.13 gives a complete description of this parameter card. (The input file is identical in structure to the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file which is described in Figure 2.10.)

2.8.2 Output

The user can choose one of two output options for program UNWRDS. The first field on the input parameter card is used to determine which option is to be taken.

If the value of this field is zero, then duplicate words on the input file are eliminated and the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file is produced. Each record on this file consists of a unique word, the number of documents that contained the word, and the total frequency with which the word occurred.

If the value of the input parameter is one, then two output files are generated. The first, the UNIQUE-WORD/CODE-NUMBER, contains one record for each unique word. Each record on this file contains a unique-word and the integer code number assigned to it. The second

Column	Program Variable	Format	Description
1	IPRM	I1	Parameter used to determine the output option. If zero, the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file is generated. If one, the UNIQUE-WORD/CODE and SORTED-WORD/DOCUMENT-NUMBER/CODE files are generated (see Figure 2.14 for a complete description of these files).
2-3	MAXWL	I2	Maximum number of characters per word, right justified in the field. This number must be identical to the second field of the parameter card for program ELDID (see Figure 2.8).

Figure 2.13

Description of Parameter Card for Program UNWRDS

file, the SORTED-WORD/DOCUMENT-NUMBER/CODE file, is identical in structure to the input file (i.e., one record for each unique word within a document) except that the word's frequency has been replaced by its unique code number. Figure 2.14 gives a complete description of the three possible files. The uses of each file will be discussed in later sections.

2.8.3 Program Description

Program UNWRDS produces the set of unique words within the document collection and assigns a unique code to each. The program compares adjacent words on the input file. The total frequency of occurrence and number of documents containing the word are accumulated as long as the current and previous word match. Whenever adjacent words do not match, a new code number is assigned to the current word and the total frequency and number of documents counters are both reset.

If option zero is specified, then whenever adjacent words do not match, the previous word, its total frequency, and the number of documents containing the word are output to the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file. Since program ELDID eliminated duplicate words within documents, there is a one-to-one correspondence between the frequency with which a word occurs on the input file and the number of documents that contain the word. The total frequency is computed by summing the frequency of the word in each document that contained it.

If option one is specified, then every record on the input file is output to the SORTED-WORD/DOCUMENT-NUMBER/CODE file after the frequency field has been replaced by the word's unique code number. Whenever

Value of Parameter IPRM	File Name	Field Number	Length (Bytes)	Program Variable	Format	Description
∅	UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY	1	2*MAXWL	IWORD	EBCDIC characters	unique word
		2	4	NDOC	Fixed Point Binary	number of documents that contained the word
		3	4	ITFQ	Fixed Point Binary	total frequency of the word
	Record Length=2*MAXWL+8					
1	UNIQUE-WORD/CODE	1	2*MAXWL	IWORD	EBCDIC characters	unique word
		2	2	NREC	Fixed Point Binary	Code number assigned to the word
	Record Length=2*MAXWL+2					
1	SORTED-WORD/DOCUMENT-NUMBER/CODE	1	2*MAXWL	IWORD	EBCDIC	word (not necessarily unique)
		2	4	NDN	Fixed Point Binary	Document Number
		3	2	KODE	Fixed Point Binary	code assigned to the word (each unique word is assigned a unique code)
	Record Length=2*MAXWL+6					

Figure 2.14

Possible Output Files from Program UNWRDS

adjacent words on the input file do not match, the previous word and its corresponding code number are output to the UNIQUE-WORD/CODE file.

Figure 2.15 presents a general flowchart of program UNWRDS.

2.9 Modifying the Set of Unique Words

Once the set of unique words has been created (output option zero), but before each word is given a unique code (output option one), the user should examine the unique words in order to make any desired changes, additions, or deletions. Output option zero of program UNWRDS will produce the set of unique words without assigning code numbers to each one. The unique words can then be examined and modified before finally running program UNWRDS with output option one, which will assign a code number to each word. This manual examination of the words by the user is why the indexing process is "semi"-automatic.

There are several reasons for examining and modifying the set of unique words before assigning code numbers:

A. There may be misspelled words that should be changed.

B. There may be words within a document that do not convey any significant information and should therefore be deleted.

C. The suffix deletion routine, within the program that extracts pertinent words from the documents (program EXTWRD), does not drop all possible suffixes; therefore, the same root word may occur with slightly different endings. All variations of the same root should be changed to the proper form of the word.

D. All documents with more words than the user allows must be examined. The user must either increase the maximum number of words per document (up to, but no larger than, 250), or

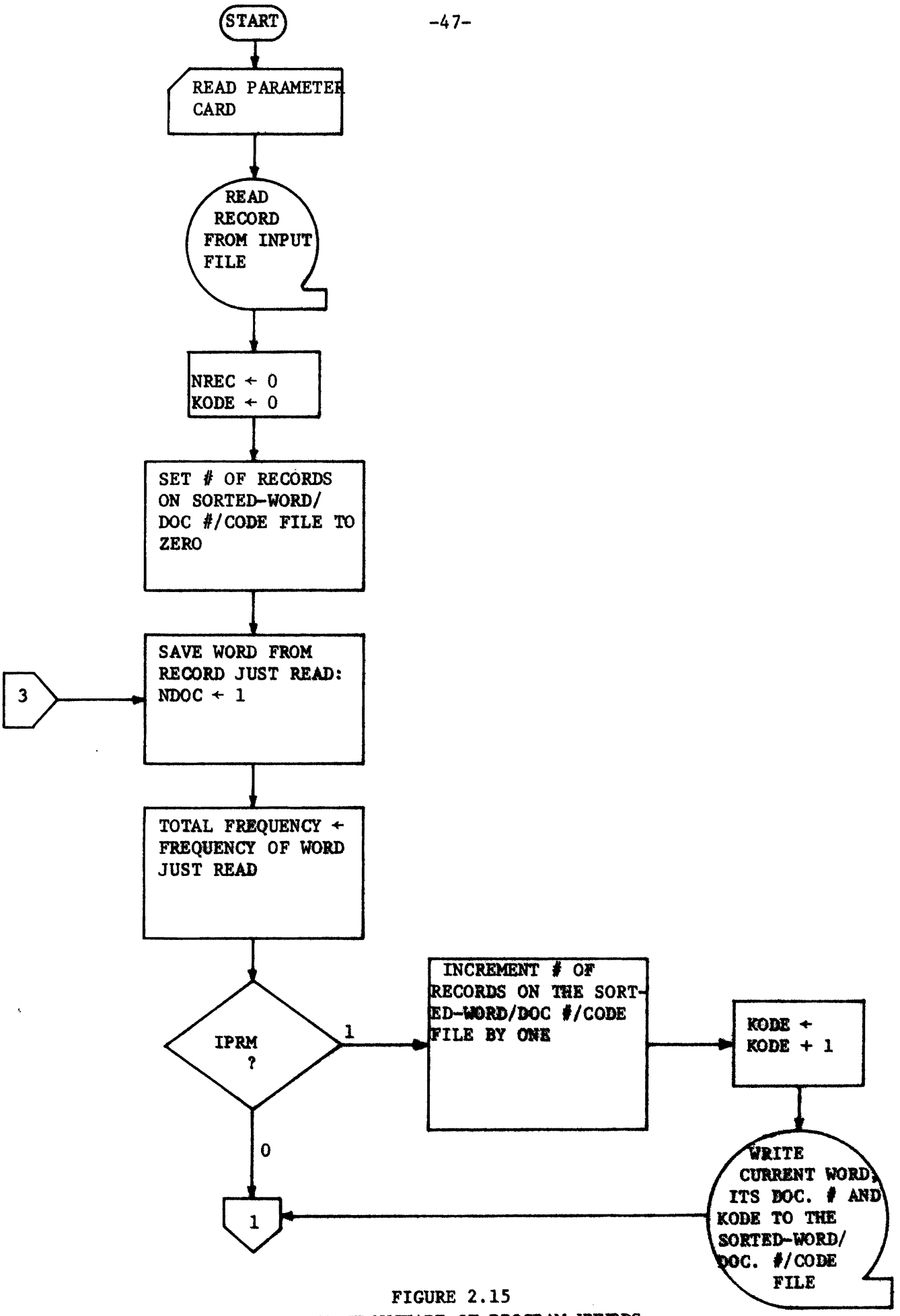


FIGURE 2.15
GENERAL FLOWCHART OF PROGRAM UNWRDS

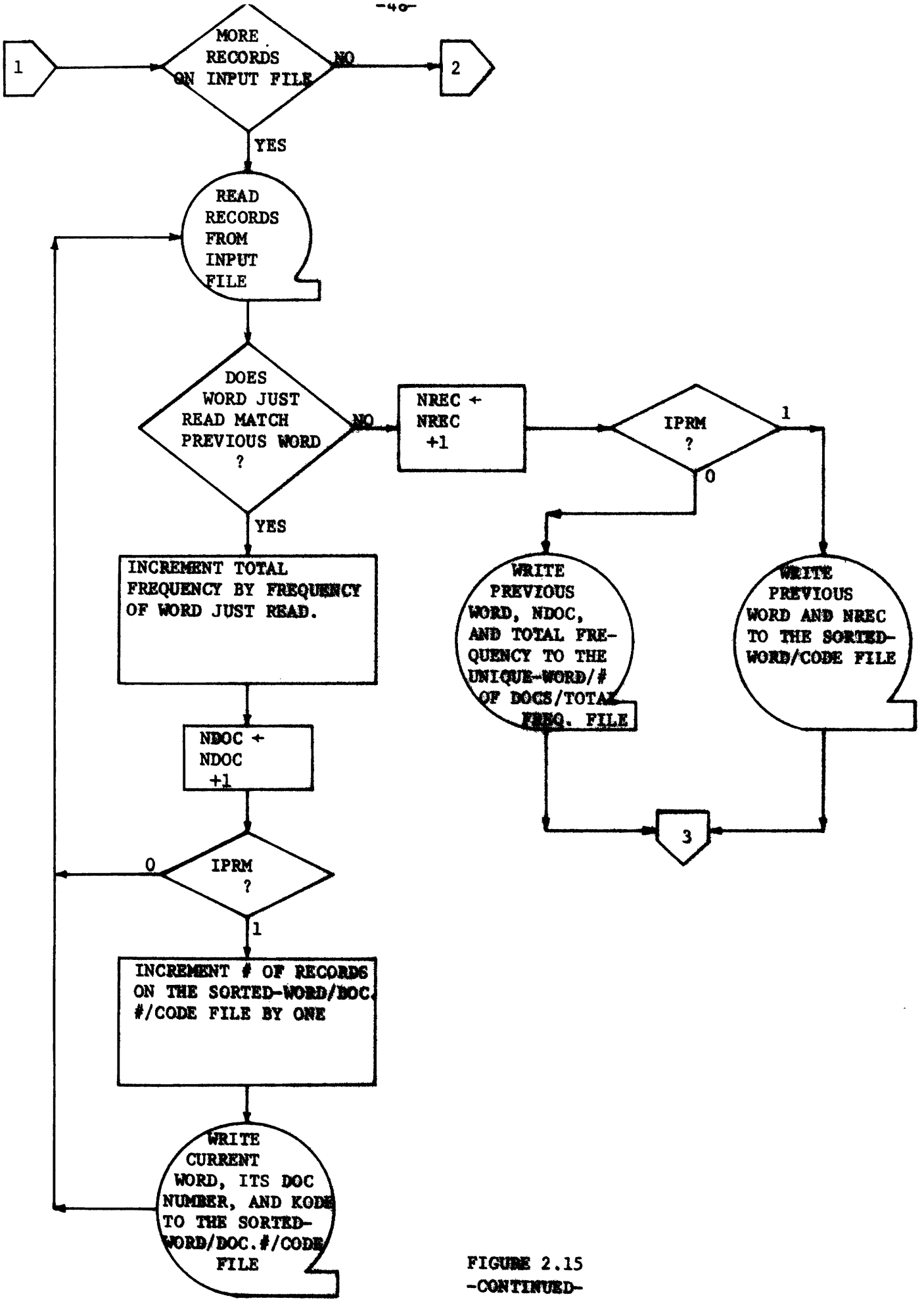


FIGURE 2.15
-CONTINUED-

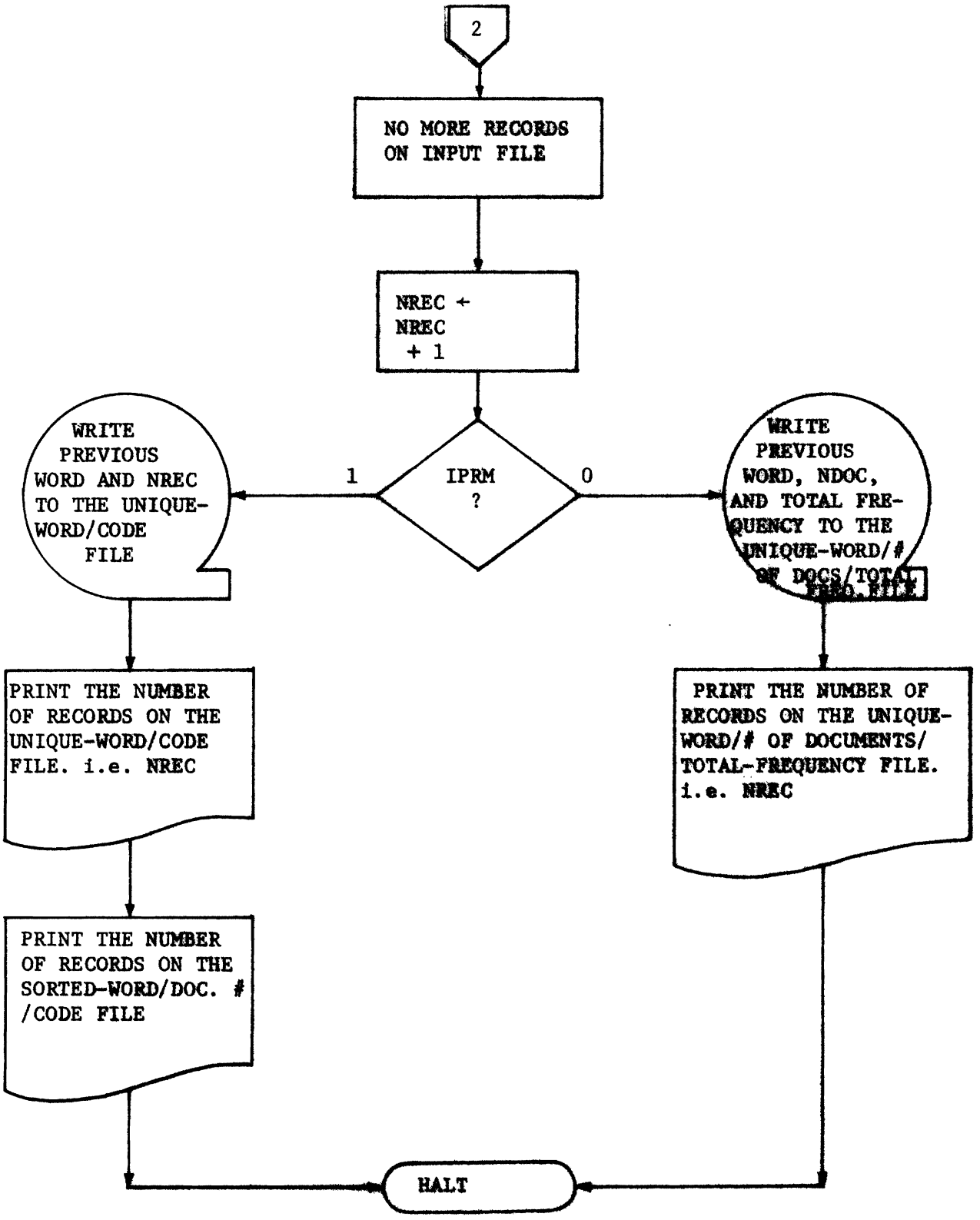


FIGURE 2.15
-CONTINUED-

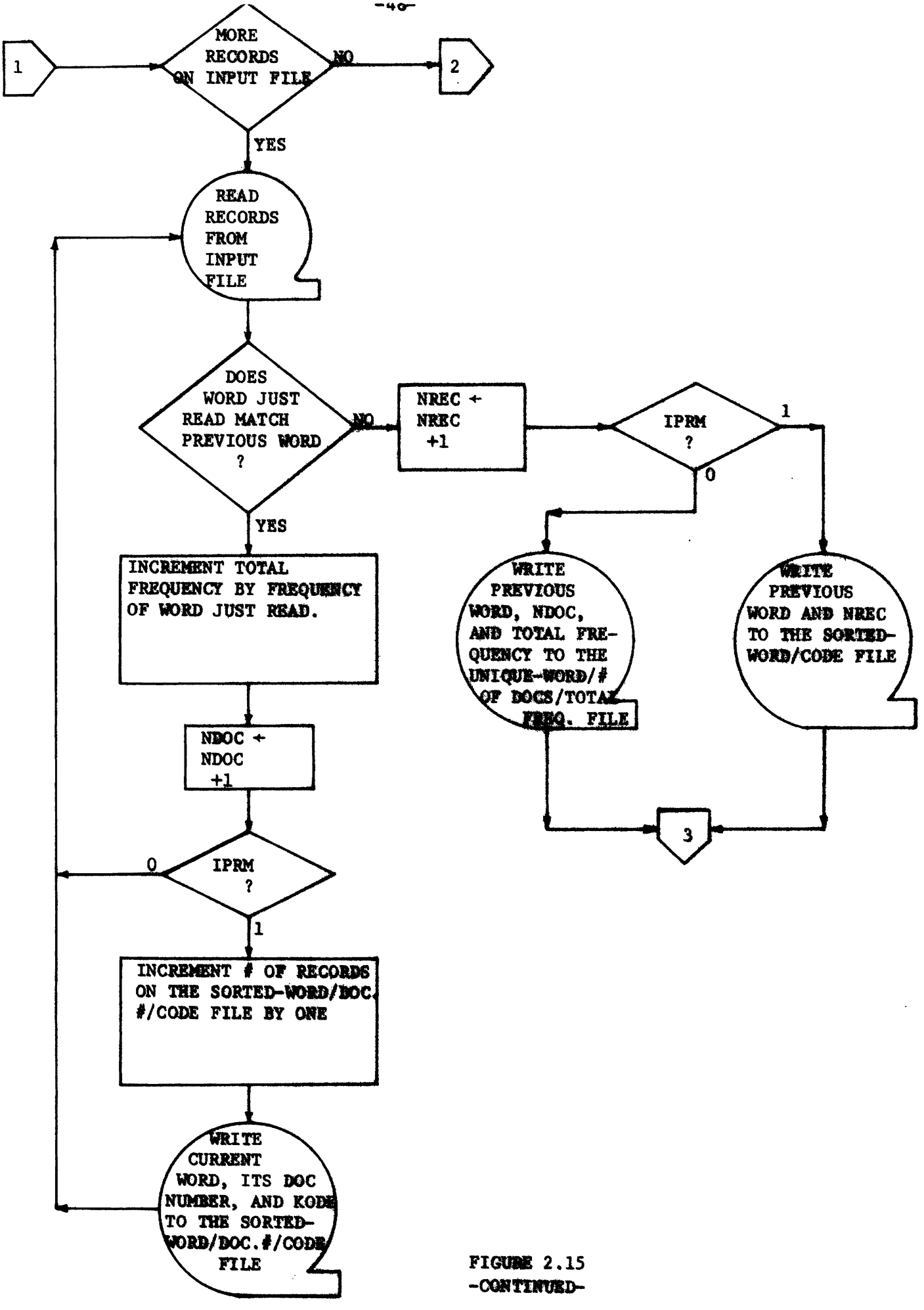


FIGURE 2.15
-CONTINUED-

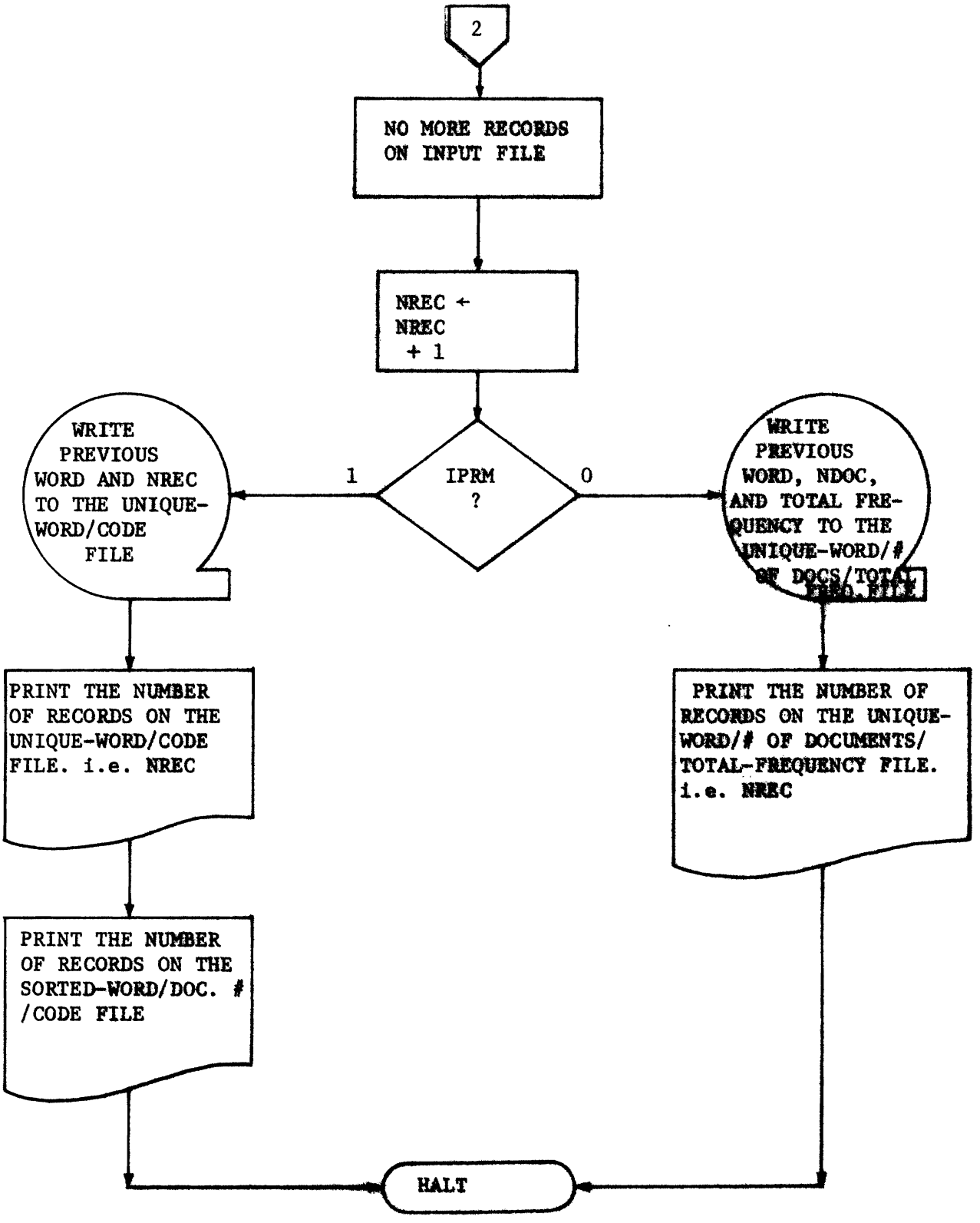


FIGURE 2.15
-CONTINUED-

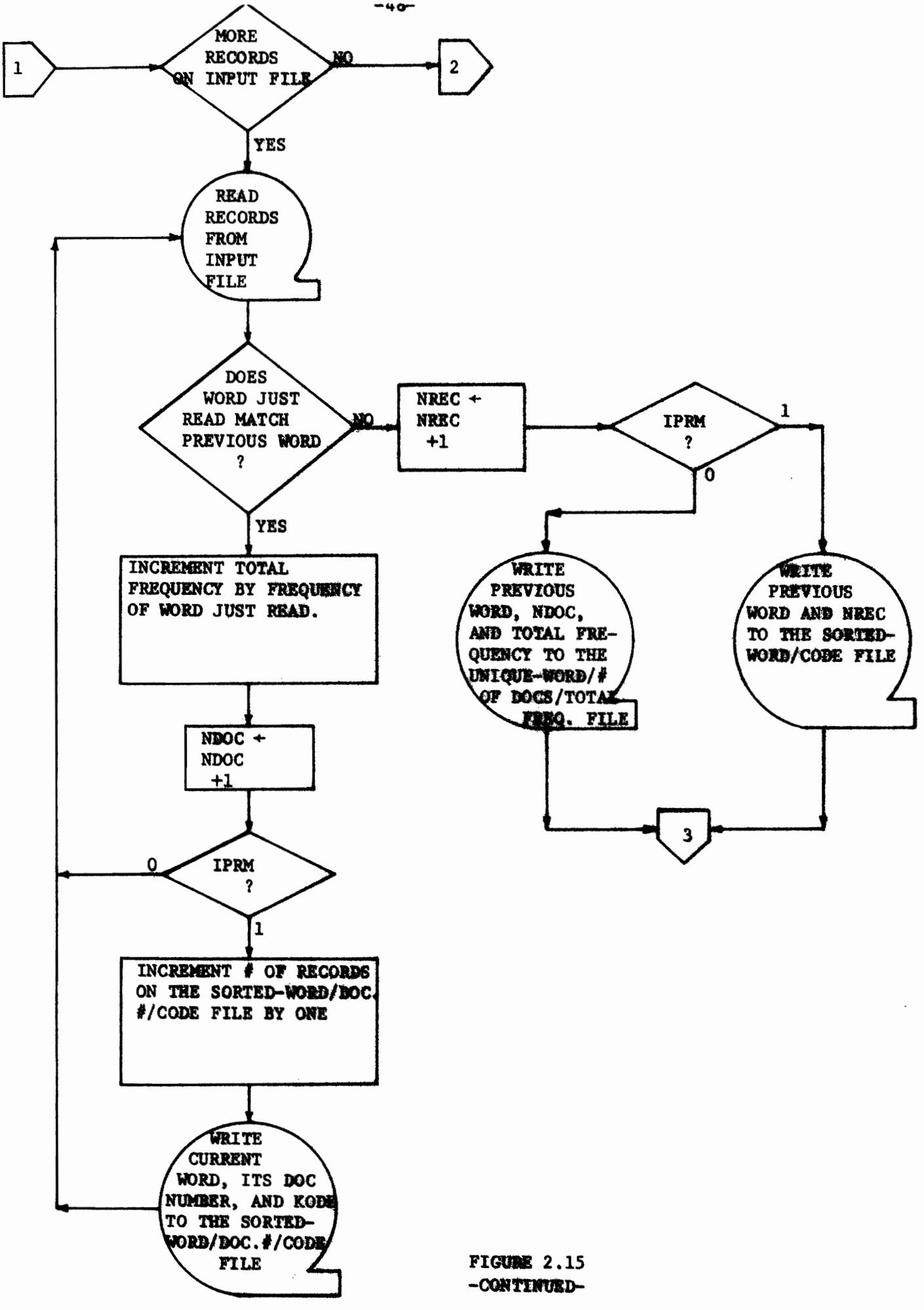


FIGURE 2.15
-CONTINUED-

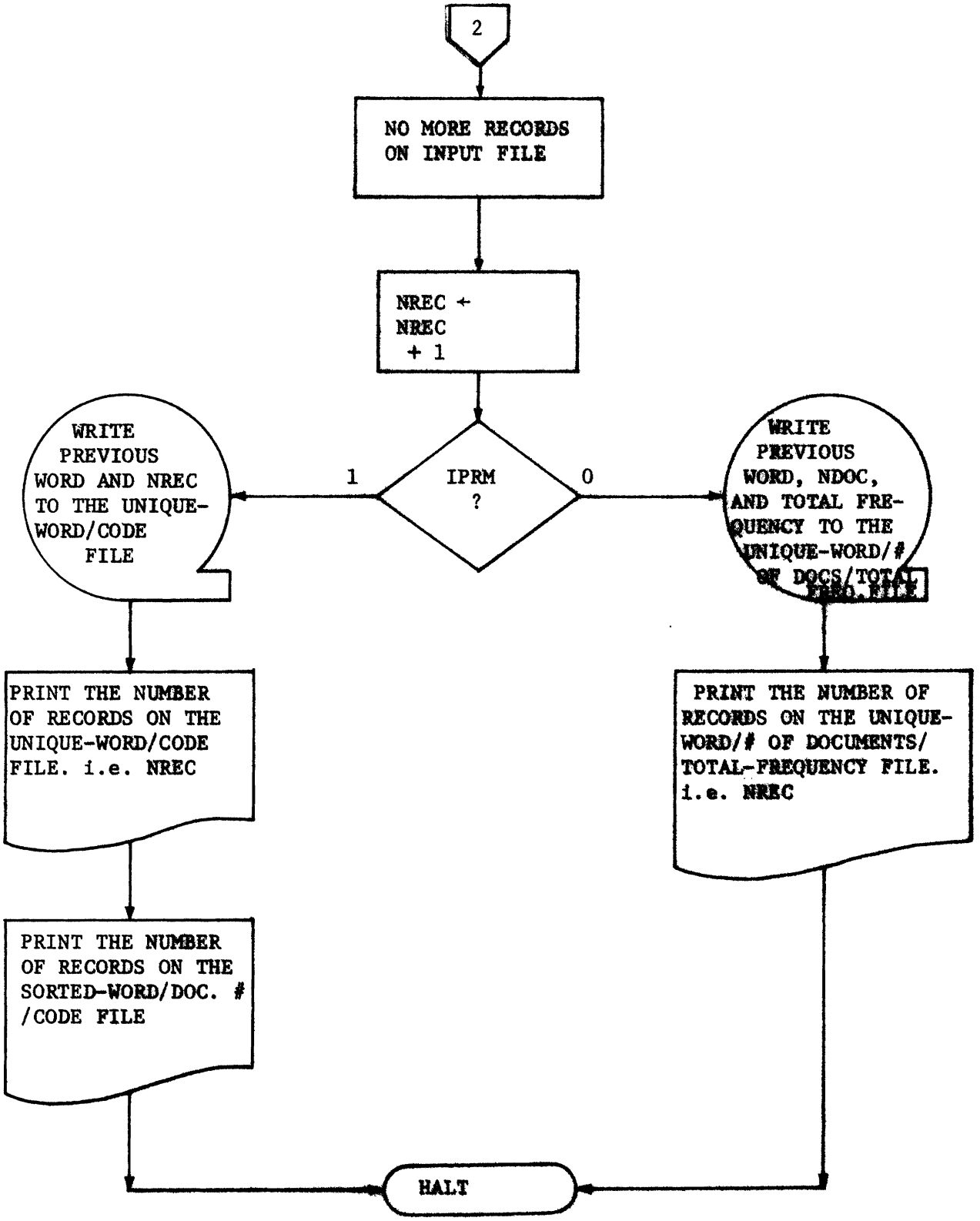


FIGURE 2.15
-CONTINUED-

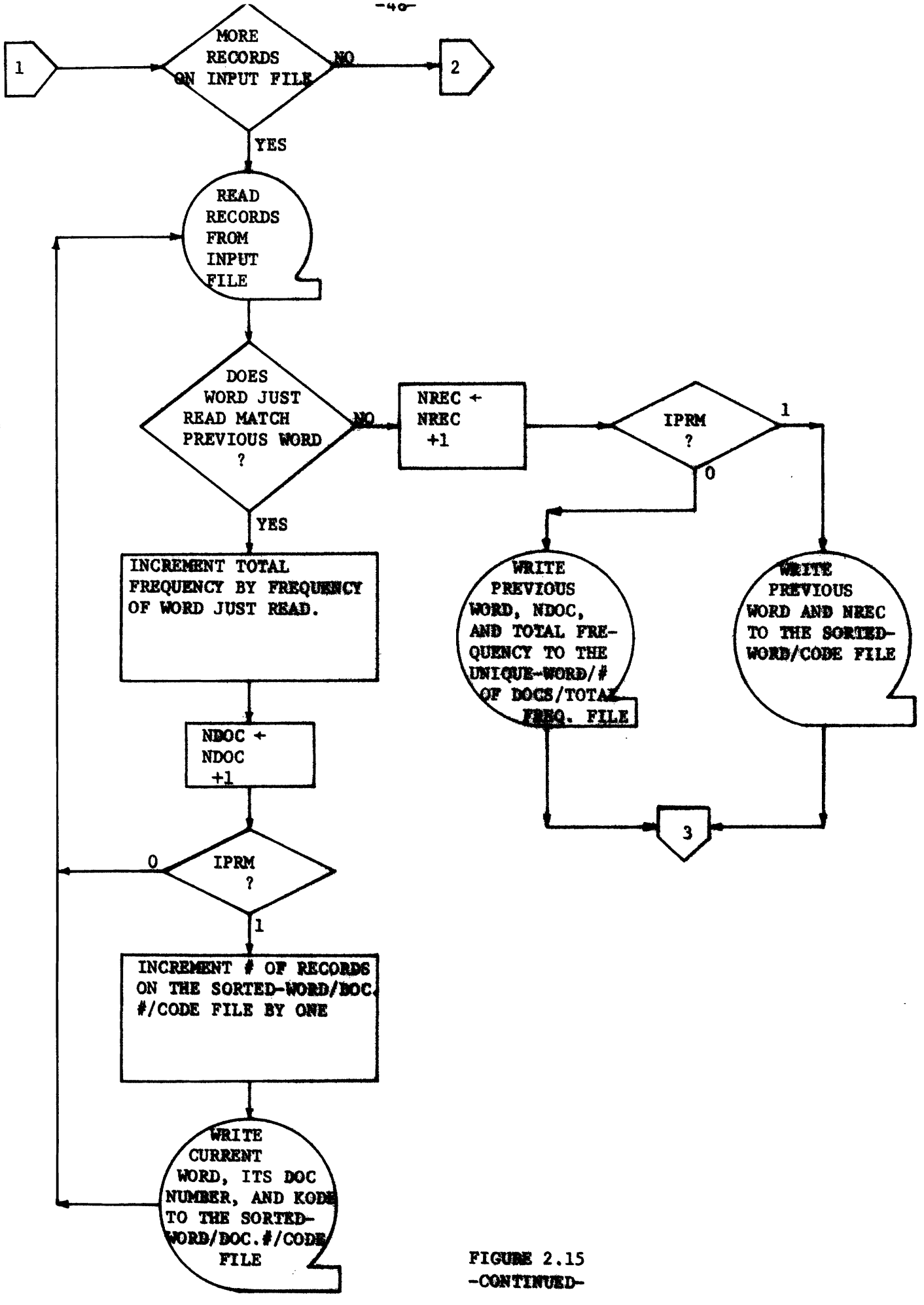


FIGURE 2.15
-CONTINUED-

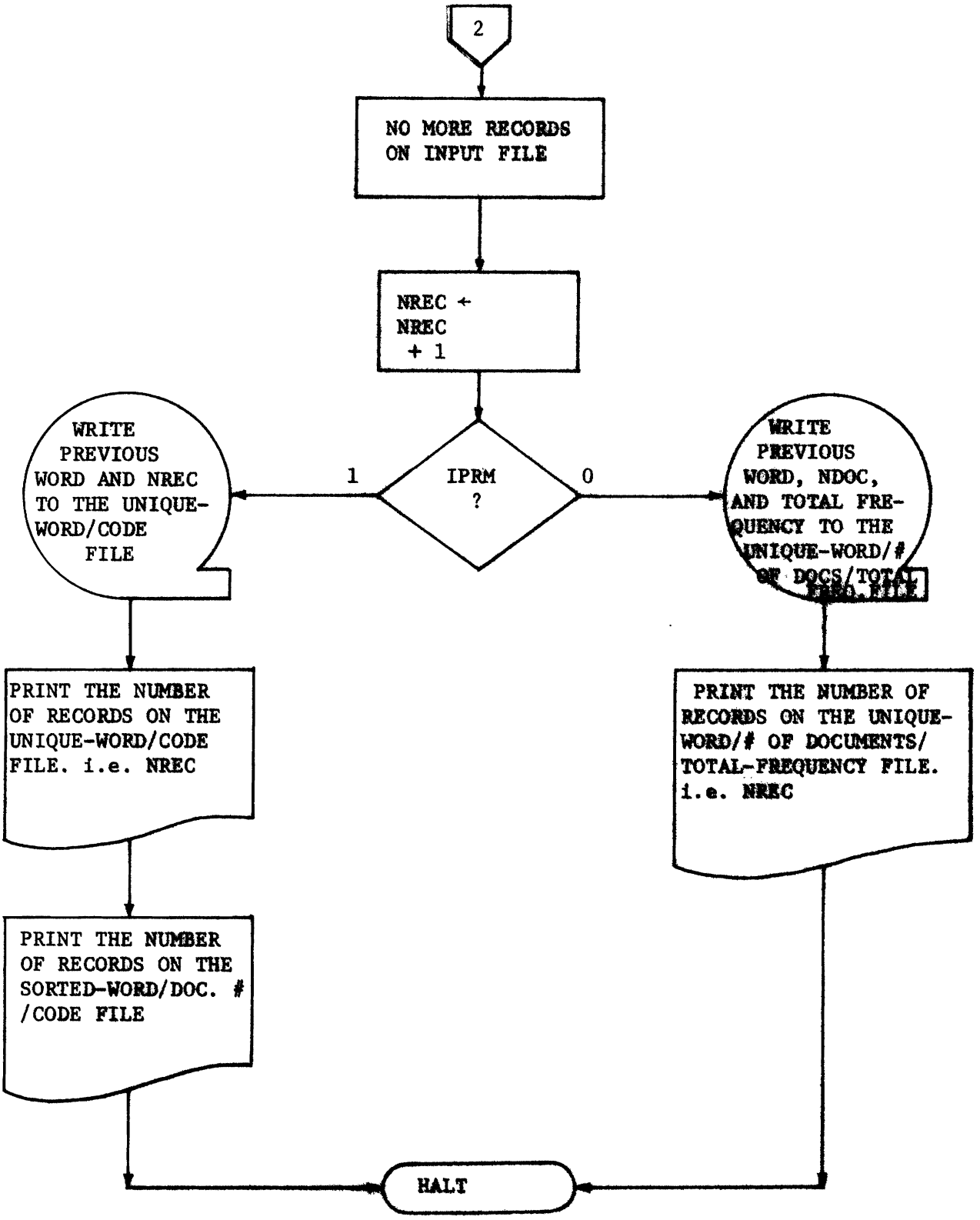


FIGURE 2.15
-CONTINUED-

delete certain words in the documents that are too large.

The following three utility programs may be used to aid the user in printing the set of unique words, deciding which modifications should be made, and actually making the additions, deletions, and changes to the set of unique words.

2.9.1 Printing the Various Files That Have Been Generated

Program PRTFIL may be used to print any one of the following files:

1. The UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY-WITHIN-DOCUMENT file
2. The SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file
3. The UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file
4. The UNIQUE-WORD/CODE file.

Input to the program is the file to be printed, an option number to tell program PRTFIL which file to expect, and the number of records to be printed. Figure 2.16 describes the required input card for this program and Figure 2.17 shows the required option number for each of the above four files and describes the corresponding listing that is generated.

2.9.2 Listing All Variations of the Same Root Word

Program ADJCMP scans the sorted words and displays adjacent words that are spelled 'similarly.' This program can be used to locate variations of the same root since all variations will be in adjacent positions on the sorted word list.

Input to program ADJCMP is a file containing sorted words and a

Column	Format	Description
1-6	I6	Document Number right justified
7-12	I6	" "
13-18	I6	" "
19-24	I6	" "
25-30	I6	" "
31-36	I6	" "
37-42	I6	" "
43-48	I6	" "
49-54	I6	" "
55-60	I6	" "
61-66	I6	" "
67-72	I6	" "
73-78	I6	" "
79 80	-	Nothing should be punched in this column. A non-blank character should be punched in column 80 if there are more document-number cards. The last document-number card of a set must have a blank in column 80.

Figure 2.21

Description of the Document-Number Card

Sorting Sequence	Character	Sorting Sequence	Character
1	A	27	'blank'
2	B	28	¢
3	C	29	.
4	D	30	<
5	E	31	(
6	F	32	+
7	G	33	|
8	H	34	&
9	I	35	!
10	J	36	$
11	K	37	*
12	L	38	)
13	M	39	;
14	N	40	¬
15	O	41	-
16	P	42	/
17	Q	43	,
18	R	44	%
19	S	45	
20	T	46	>
21	U	47	?
22	V	48	:
23	W	49	#
24	X	50	@
25	Y	51	'
26	Z	52	=
		53	"

Figure 2.22

Sorting Sequence to Be Used in "Alphabetizing" Words

4) An ERASE command with a non-blank column 80. (ERASE must be performed on all occurrences of the word(s).) Care should be taken in using the ERASE command. If the second word in the operand field is incorrect, all words on the input file from the first word in the operand field until the incorrect second word will be deleted. If the second word is not on the input file, all words from the first until the position where the misspelled word would occur in the input file would be deleted. A misspelled second word can therefore cause a significant loss of data.

5) An ADD command with a blank column 80. (ADD can only be performed on specific documents.)

6) Trying to add a word that is already in the file.

7) The document numbers are not in ascending order on the document-number card(s).

8) Trying to DELETE, CHANGE, or ERASE words that are not on the input file.

If any of the above conditions occur, an appropriate error message will be printed. These messages are self explanatory and the user should change the card(s) that caused the error and rerun the program. Before the program can be restarted, however, the MODIFIED-SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file must be sorted on the word and document number. (Section 2.7 describes the required sort.)

2.9.4 Effectively Using the Utilities to Make Modifications to the Unique Words

As stated previously, before assigning code numbers to the set of unique words, the user will probably want to examine and modify these

words. The utility programs, previously described, can be used as 'tools' to not only help the user decide what modifications should be made, but also to physically modify the SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file.

The user has previously input to program EXTWRD the maximum number of pertinent words allowed per document, and the program produced a list of the documents that were too large. This list contains the document number and the total number of words in each document containing more words than the user allows.

Program ADJCMP can be run, either before or after program UNWRDS, in order to produce a list of similar words. If the user waits until after running program UNWRDS to execute the adjacent word comparison routine, ADJCMP, then he can use the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file as input. In this case, the number of documents containing each word and each word's total frequency will be printed along with the groups of similar words. If the user wishes to save time by running the adjacent word compare routine before he runs UNWRDS, he must use the SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file as input, and the groups of similar words will be printed without any associated statistics. The advantage of obtaining the statistics (number of documents containing the word and total frequency) along with the groups of similar words is that they may be used to determine a word's relative importance.

The user also has the option of listing several files that have been created. These listings can be used to help the user decide what modifications should be made. Using options 3, 2, and 2 of program PRIFIL, the user can list the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-

FREQUENCY, SORTED-WORD/DOCUMENT-NUMBER FREQUENCY, and UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY files respectively. The UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file can also be sorted by either the number of documents containing the word or the total frequency fields and listed with option 3 of program PRTFIL. Assuming that the user has obtained these four listings, he should consider the following procedures in deciding what modifications should be made to the SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file:

1) The listing of the UNIQUE-WORDS/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file should be examined thoroughly. This list can be used to easily locate groups of words, with no apparent information content, that can be ERASED or DELETED. Misspelled words must also be located and CHANGED to their correct spelling. Since no document numbers appear on this listing, any modifications made in conjunction with this listing must be made to all occurrences of the word (i.e., column 80 on the modification command must be blank).

2) If the UNIQUE-WORDS/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file is sorted by either the number of documents containing the word or the total frequency fields and listed by option number 3 of program PRTFIL, then the user may utilize this listing to determine the relative importance of certain words. Very high and very low frequency words should be thoroughly examined since they have a significant effect upon the resulting classification. The user should DELETE any nonsense words and CHANGE all misspellings. Again, any modifications made by using this listing must be to all occurrences of the word.

3) The list of similar words can be used to determine CHANGES that must be made. All variations of the same root word should be CHANGED

to the proper spelling of the root. In order to insure consistancy, all CHANGES should be made to every occurrence of the word.

4) In any of the above three procedures, if the user needs to know what documents mentioned a particular word, then he should consult the listing of the SORTED-WORDS/DOCUMENT-NUMBER/FREQUENCY file. This listing shows the numbers of all documents that mention each word and the user can include the document numbers he wishes to modify on the document-number card(s) immediately following the corresponding modification command card.

5) The listings of the document numbers that contain more words than the user allows (output from program ELDID) and the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file can be used to reduce the number of words in all documents that are too large. The user can examine the words in each document that is too large and DELETE or CHANGE words in order to reduce the total number of words in each document within the user's specification. If it is impossible to reduce the size of all documents to within the maximum, then the maximum must be redefined and given a higher value. (Note: the maximum cannot be set greater than 250)

The user is not restricted to the procedures and utilities described in this section. These are 'tools' to be used at his discretion. The user has the option of writing his own utility programs and designing procedures that will aid his decision as to what modifications must be made to the SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file.

By whatever means he chooses, the user must decide upon the needed modifications and punch the corresponding modification command cards. These cards must be arranged to correspond to the order in which the

words to be modified appear on the listing of the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file.

After running program UTILKS with the newly created modification cards as input, the user must "re-cyle" through several steps before he can assign code numbers to each unique word and finally create surrogates for each document. "Re-cycling" is necessary because some of the modifications may have created duplicate words within some documents and these must be eliminated. The following steps must be taken in order to "re-cycle":

1) Sort the MODIFIED-SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file, which is output from program UTILKS, by document number and word. See Section 2.5 for a description of the required sort. (The sort of Section 2.5 can be used even though the records on the previous input file to the sort were two bytes shorter.)

2) Run program ELDID, specifying input option zero. (See Section 2.6.) The output of the sort of Step 1 above is to be used as input. The user may also redefine the maximum number of words allowed per documents when running ELDID.

3) The output file from ELDID should then be sorted by word, document-number. Section 2.7 describes this sort.

4) Program UNWRDS must now be executed with the output of the sort in Step 3 above used as the input file. If the user feels that more modifications may be necessary, he can specify output option zero and re-cycle again. If the user thinks that he has sufficiently "cleaned-up" the set of unique words and is ready to create the document surrogates, then he should terminate the re-cycle process and specify output option one.

The user may "re-cycle" as many times as necessary until he is satisfied that the set of unique words are relatively free of misspellings, nonsense words (that convey little information), and that most variations of the root words have been changed to a consistent form. The documents with more words than the user allows should have been examined, and either the total number of words in each reduced or a new maximum ($\leq$ 250) defined to cover the largest document. When the user has sufficiently "cleaned-up" the set of unique words, re-cycling is terminated and codes may be assigned to each word by running program UNWRDS with output option one (see Step 4 above). This output option will cause the UNIQUE-WORD/CODE and SORTED-WORD/DOCUMENT-NUMBER/CODE files to be generated; the latter file will be used to create the surrogates for each document.

2.10 Creating the Document Surrogates

After the user is satisfied that the unique words have been "cleaned-up" by "re-cycling" with the utilities and procedures described in the last section, he is ready to create a surrogate for each document. The final step in the "re-cycling" procedure was to run program UNWRDS with output option one, thus producing the UNIQUE-WORD/CODE and SORTED-WORD/DOCUMENT-NUMBER/CODE files.

The user should list the UNIQUE-WORD/CODE file with option 4 of the utility program PRTFIL. This listing, which may be used as a reference, contains the set of unique words and their respective code numbers.

The SORTED-WORD/DOCUMENT-NUMBER/CODE file is used to create a surrogate for each document. As shown in Figure 2.14, each record

on this file contains a word, the number of the document that it appeared in, and the word's code number. If the word occurred in several documents, then there is a record for each document that mentioned the word. (Each of these records will have identical word and code number fields; the document number fields will correspond to the numbers of the documents that contained the word.) Since the SORTED-WORD/DOCUMENT-NUMBER/CODE file is in "alphabetical" order, it must first be sorted by document before it can be used to create the document surrogates.

2.10.1 Sorting the SORTED-WORD/DOCUMENT-NUMBER/CODE File by Document, Code

The SORTED-WORD/DOCUMENT-NUMBER/CODE file must be input to a sort routine that uses the document-number as the major sort field and the code number as the minor field. Figure 2.23 summarizes the sort field parameters that must also be input to the sort routine. Output from this sort will be the DOCUMENT-NUMBER/CODE/WORD file. The structure of this output file is identical to the input file, except that the words are now in "alphabetical" order by document. After completing the sort, program DOCSUR may be run to create the surrogates for each document.

2.10.2 Input to DOCSUR

Input to program DOCSUR is the DOCUMENT-NUMBER/CODE/WORD file just created and a parameter card containing the number of documents in the collection, maximum number of words per document, and the maximum number of characters per word. The structure of the DOCUMENT-

Field Number	Type*	Length (Bytes)	Description
1	FI,A	4	Document Number
2	FI,A	2	Code Number

*FI denotes fixed point integer; A denotes ascending sequence

Figure 2.23

Description of Sort Field Parameters
Required to Sort by Document Number, Code

NUMBER/CODE/WORD file is identical to the SORTED-WORD/DOCUMENT-NUMBER/CODE file (see Figure 2.14) and the parameter card is described in Figure 2.24.

If the user does not recall the exact number of documents in his collection, he should consult the printed output from program ELDID.

The maximum number of words allowed per document has been discussed with respect to program ELDID (see Section 2.6.1). That program produced a list of the documents that were too large and at this point in the indexing process the user must have either reduced the size of those documents (using the utilities and procedures outlined in Section 2.9.4), or redefined his maximum to correspond to the largest document. Program DOCSUR will drop words from any document containing more than the maximum number of words read from the input card; therefore, the user should define this maximum value to correspond to the largest document in order to avoid any loss of information. (Note that the maximum must be less than or equal to 250.)

2.10.3 Output from DOCSUR

Output from program DOCSUR is the document surrogate file, SURROG. Each record on this file corresponds to a document surrogate. (See Figure 2.25 for a complete file description.)

2.10.4 Program Description

Program DOCSUR reads the DOCUMENT-NUMBER/CODE/WORD file and accumulates the codes that correspond to the descriptors assigned to each document. When the document number in the current record does not match the previous document's number, then the codes accumulated up

Column	Program Variable	Format	Description
1-6	NDOC	I6	Number of documents in the user's collection, right justified in the field. This value may be obtained from the printout from program ELDID.
7-9	MAXKS	I3	Maximum number of words allowed per document, right justified in this field. Any document with more than this maximum number of words will have its extra words dropped; therefore, the user should insure that this value corresponds to the size of his largest document.
10-11	MAXWL	I2	Maximum number of characters per word, right justified in this field. This number must be consistant throughout all of the Semi-Automatic Indexing routines.

Figure 2.24

Description of the Parameter Card for Program DOCSUR

Field Number	Length (Bytes)	Program Variable	Format	Description
1	4	IPREVD	Fixed Point Binary	Document Number
2	2	NDKY	Fixed Point Binary	Number of codes (words) in this surrogate (must be $\leq$ 250)
3	4	LEN	Fixed Point Binary	Length of cell (must be the actual number of surrogates - first record only)
4	2	KCL	Fixed Point Binary	Terminal cell flag (not used in program DOCSUR)
5	2	LEV	Fixed Point Binary	Level in which terminal cell occurred (not used in probram DOCSUR)
6	2	INODE	Fixed Point Binary	Node number of terminal cell (not used in program DOCSUR)
7	2	KYSUR(1)	Fixed Point Binary	First code in surrogate
8	2	KYSUR(2)	Fixed Point Binary	Second code in surrogate
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
6+NDKY	2	KYSUR (NDKY)	Fixed Point Binary	Last code in surrogate

Record Length = (NDKY*2+16)

Figure 2.25

Description of the Document Surrogate File, SURROG

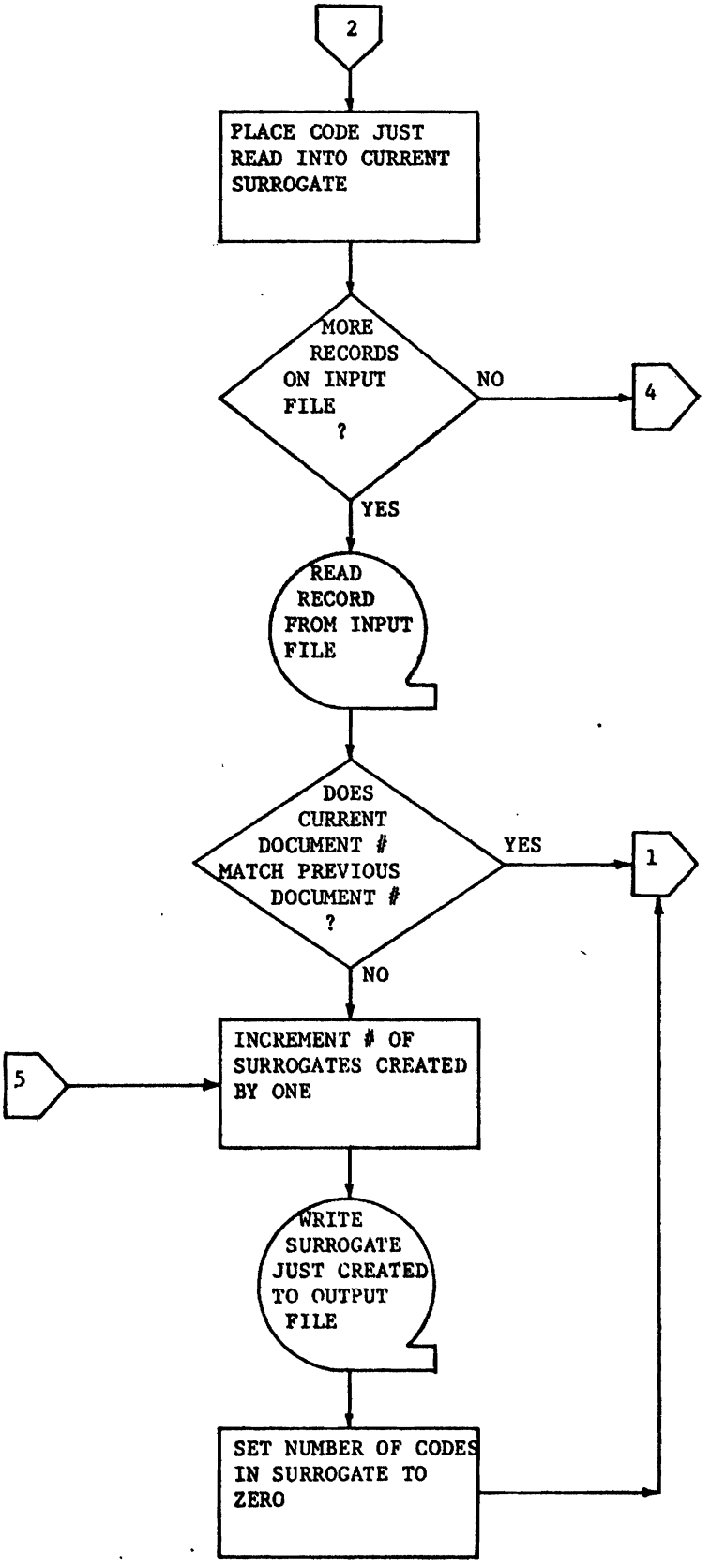


FIGURE 2.26
-CONTINUED-

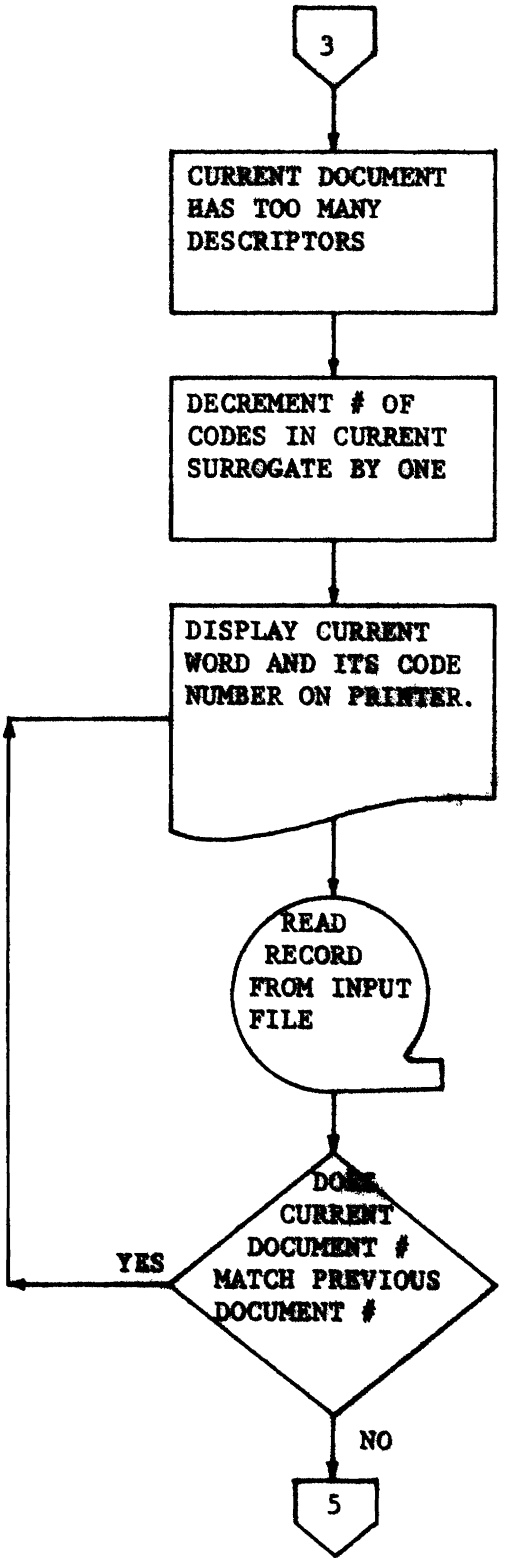


FIGURE 2.26
-CONTINUED-

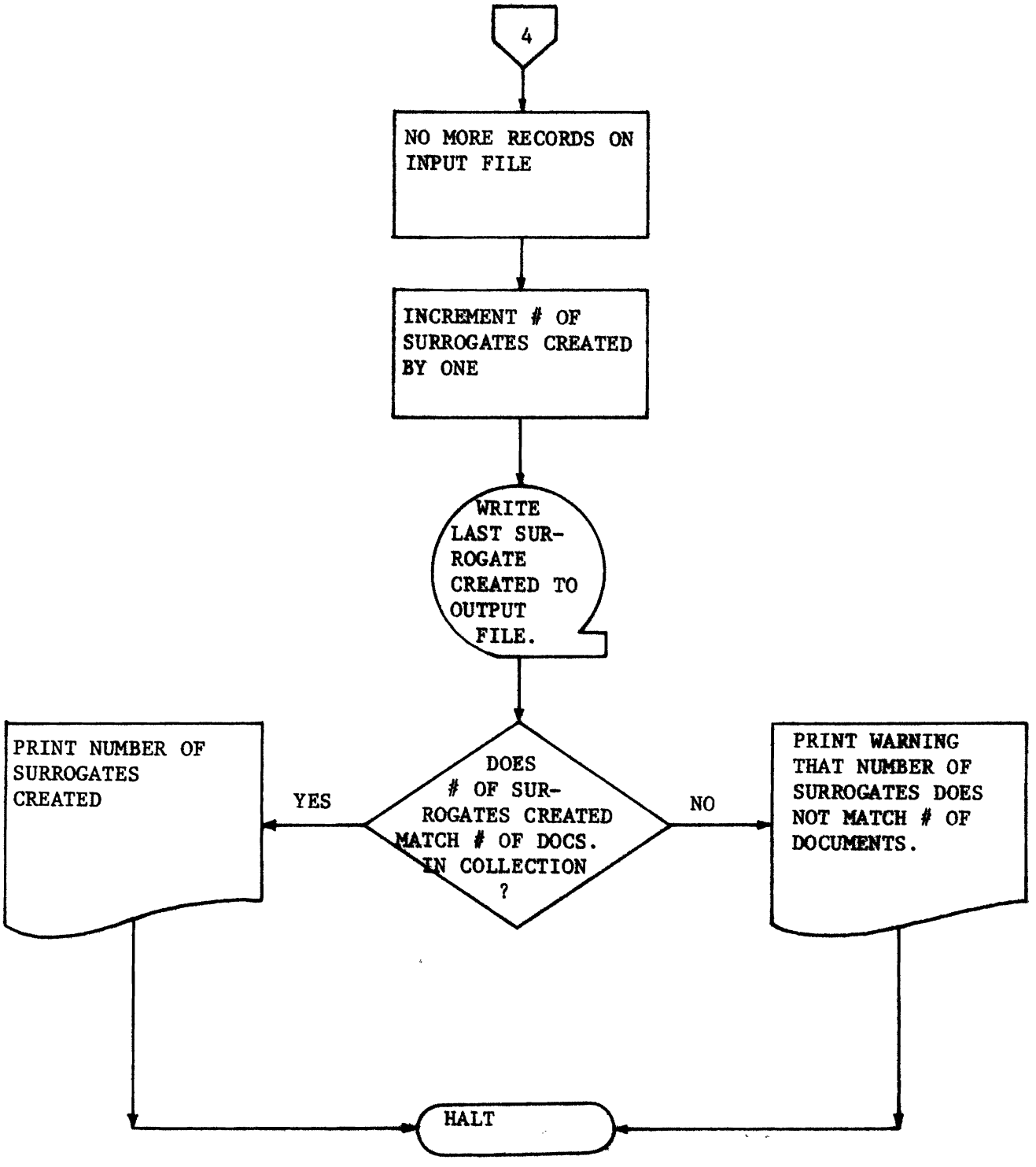


FIGURE 2.26
-CONTINUED-

Creating the document surrogate file, SURROG, completes the Semi-Automatic Indexing Routines. This file, which contains a surrogate for each document, is the main input file to the Automatic Classification Routines which are discussed in the following chapter. These routines will re-arrange the document surrogates into cells, each cell being a classification and containing only surrogates of similar documents.

2.11 Summary and Examples of the Semi-Automatic Indexing Procedure

This section can be used as a reference when running the Semi-Automatic Indexing Routines. A set of steps are provided that must be followed in order to index the user's document collection, i.e., transform each source document into a surrogate (on file SURROG) for input to the Automatic Classification Routines.

Each of the steps presented in this section either describes a program that must be run or a decision that the user must make in order to proceed with the Semi-Automatic Indexing process. The purpose, input and output of each step are given. The inputs are data cards and/or a file created in a previous step. (All files are sequential and may be either disk or tape volumes.) Outputs are printouts and/or file(s) to be input to a later step(s). References are given in each step to the section within this chapter that describes the step and to the figures that describe the input cards. If a step should fail, due to the abnormal termination of its program, then the failing step must be restarted. (Restarting from the beginning of a step can easily be done by providing the required input, output, and running the required program.) Each step is also illustrated with

an example from the Semi-Automatic Indexing of a data base obtained from the Foreign Broadcast Information Service. The data base contains the complete text of 1669 messages (documents) that were broadcast in 1971 and deal with world-wide events--political, military, social, and economic. A typical message (document) consists of 200-300 words. A few of the messages are considerably shorter and some are quite long (i.e., 1000 words or more). The Semi-Automatic Indexing Routines and Automatic Classification Routines were run on the Moore School's Spectra 70 and the messages (documents) of the FBIS data base were indexed and automatically classified. Each step illustrated in this section shows the deck setup used on the Spectra 70 and any output generated from the semi-automatic indexing of the FBIS data base. The deck setup consists of:

1) log-on to computer
2) definition of input and output files
3) execution command to run the program
4) the program's corresponding data card(s)
5) log-off from computer.

The printouts contain run time error messages and file statistics (i.e., number of output records). All information shown in the examples pertain to the indexing of the FBIS data base, run on the Spectra 70. This same data base is also used to illustrate the Automatic Classification Routines described in the next chapter. In order to further clarify the steps presented in this section, the user will want to refer to Figures 2.27 and 2.28.

Figure 2.27 summarizes all of the files needed for Semi-Automatic Indexing. The following information is given for each file: its name

and reference used in Figure 2.28, the program that creates the file, the program(s) that use the file as input, the number of the figure that gives a complete description of the file, the record length in bytes (for variable length records, the maximum length is given), the record type (variable or fixed length records, and a brief description of the file's contents.

Except for the Standard Input File, which contains the user's documents in a fixed format, and the SURROG file, which contains the surrogates for each document, a standard file naming convention is used throughtout this chapter. The file names have the following form: NAMEAA/NAMEBB/NAMECC, where 'NAMEXX' is the name of a field within the file's record. If the file's records have three(two) fields, then the file name has three (two) NAMEXX's. For example, the file that contains the unique words within each document has the following three fields in every record: (1) the unique word, (2) the document number, (3) the frequency of occurrence within the document. The name of this file is the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file.

Figure 2.28 gives a generalized flowchart of the Semi-Automatic Indexing Routines (Steps). The flow of processing is shown down the center of each page and is represented by solid lines connecting each processing box. (Each processing box corresponds to a step.) All inputs to each step and outputs from each step are given on the left and right respectively of the step. These inputs and outputs are represented by the horizontal dashed lines entering and leaving each processing step. If a file or listing is output from one step and input to a later step, then it will appear on the right of the step that

File Name[2]	Output From	Input To	File Description Given in Figure	Record Length (Bytes)[1]	Record Type	Description
Standard Input File (reference #1)	user written program	EXTWRD	2.0	4096 (maximum)	V	This file contains the titles, abstracts, full text, keywords, or any combination of these from the user's documents. This information is placed on the Standard Input File in a fixed format by a user written program.
WORD/DOCUMENT-NUMBER (reference #2)	EXTWRD	Sort by doc. #, word	2.4	44	F	Each record on this file contains a pertinent word extracted from a document and the document's number. The words are grouped by document.
SORTED-DOCUMENT-NUMBER/WORD (reference #3)	Sort by doc. #, word	ELDID	2.9	44	F	Each record on this file is identical in structure to the WORD/DOCUMENT-NUMBER file except that they have been sorted into "alphabetical" order by document.
UNIQUE-WORD/ DOCUMENT-NUMBER/ FREQUENCY (reference #4)	ELDID	Sort by word, doc. # PRTFIL	2.10	46	F	Each record on this file contains a unique word that appeared within a document, the document's number, and the frequency of the word. The words are in "alphabetical" order by document.
SORTED-WORD/ DOCUMENT-NUMBER/ FREQUENCY (reference #5)	Sort by word, doc. #	UNWRDS UTILKS ADJCMP PRTFIL	2.10[4]	46	F	Each record on this file is identical in structure to the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file except that the words are all in "alphabetical" order. If a word appears in several documents, then there is a record for each document that mentions the word.
UNIQUE-WORD/ NUMBER-OF-DOCUMENTS/ TOTAL-FREQUENCY (reference #6)	UNWRDS	PRTFIL ADJCMP	2.14	48	F	Each record on this file contains a unique word from the entire collection of documents, the number of documents that mention the word, and the word's total frequency. The words are in "alphabetical" order.

Figure 2.27 Summary of the Semi-Automatic Indexing Files

File Name[2]	Output From	Input To	File Description Given in Figure	Record Length (Bytes)[1]	Record Type	Description
MODIFIED-SORTED-WORD/ DOCUMENT-NUMBER/ FREQUENCY (reference #7)	UTILKS	Sort by doc. #, word	2.1Ø[4]	46	F	Each record on this file is identical in structure to the SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file. This file contains all of the modifications that the user has made to the words (i.e., ADD, DELETE, CHANGE, and ERASE).
SORTED-DOCUMENT-NUMBER/WORD/FREQUENCY (reference #3A)	Sort by doc. #, word	ELDID	2.9	46	F	This file is the MODIFIED-SORTED-WORD/DOCUMENT-NUMBER/ FREQUENCY sorted such that the words are in "alphabetical" order by document.
SORTED-WORD/ DOCUMENT-NUMBER/ CODE (reference #8)	UNWRDS	Sort by doc. #, code	2.14	46	F	Each record on this file contains a word, the number of the document that mentioned the word, and the word's code. If a word occurred in several documents, then there is a record for each document that mentioned the word. The words are in "alphabetical" order.
DOCUMENT-NUMBER/ CODE/WORD (reference #1Ø	Sort by doc. #, code	DOCSUR	2.14[3]	46	F	Each record on this file is identical in structure to the SORTED-WORD/DOCUMENT-NUMBER/CODE file. The words are now in "alphabetical" order by document.
SURROG	DOCSUR	Automatic Classification Routines	2.25	516 (maximum)	F	Each record on this file contains a document surrogate. These are ordered by document number.
UNIQUE-WORD/CODE (reference #9)	UNWRDS	PRTFIL	2.14	42	F	This file contains the unique words of the entire document collection in "alphabetical" order. Along with each word is its code number.

[1]Each record length was computed assuming that 2Ø was used as the maximum number of characters per word (MAXWL). The records are written in a binary (unformatted) mode; hence, the user must be aware of any control information that is suffixed to each unformatted record by the operating system, thus increasing the length of each record.

[2]Along with the file name is a reference number which corresponds to the reference numbers used in Figure 2.28.

[3]Figure 2.14 gives the description of an identical file: SORTED-WORD/DOCUMENT-NUMBER/CODE.

[4]Figure 2.1Ø gives the description of an identical file: UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY.

Figure 2.27, continued

generates it and on the left of the step(s) that require it as input. Reference numbers are given to the files and letters to the listings. (All files are sequential and may be either tape or disk.)

For example, program ELDID produces an output file (reference number 4), UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY, and a listing (reference letter E) of the numbers of the documents that were too large. Both of these are on the output (right) side of the ELDID processing step. The UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file is later input to a sort and program PRIFIL. The listing is input to the step that requires the user to examine listings in order to decide what, if any, modifications he wishes to make to the unique words of the document collection. In these cases the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file and the listing of documents that are too large appear on the input (left) side of the corresponding processing steps.

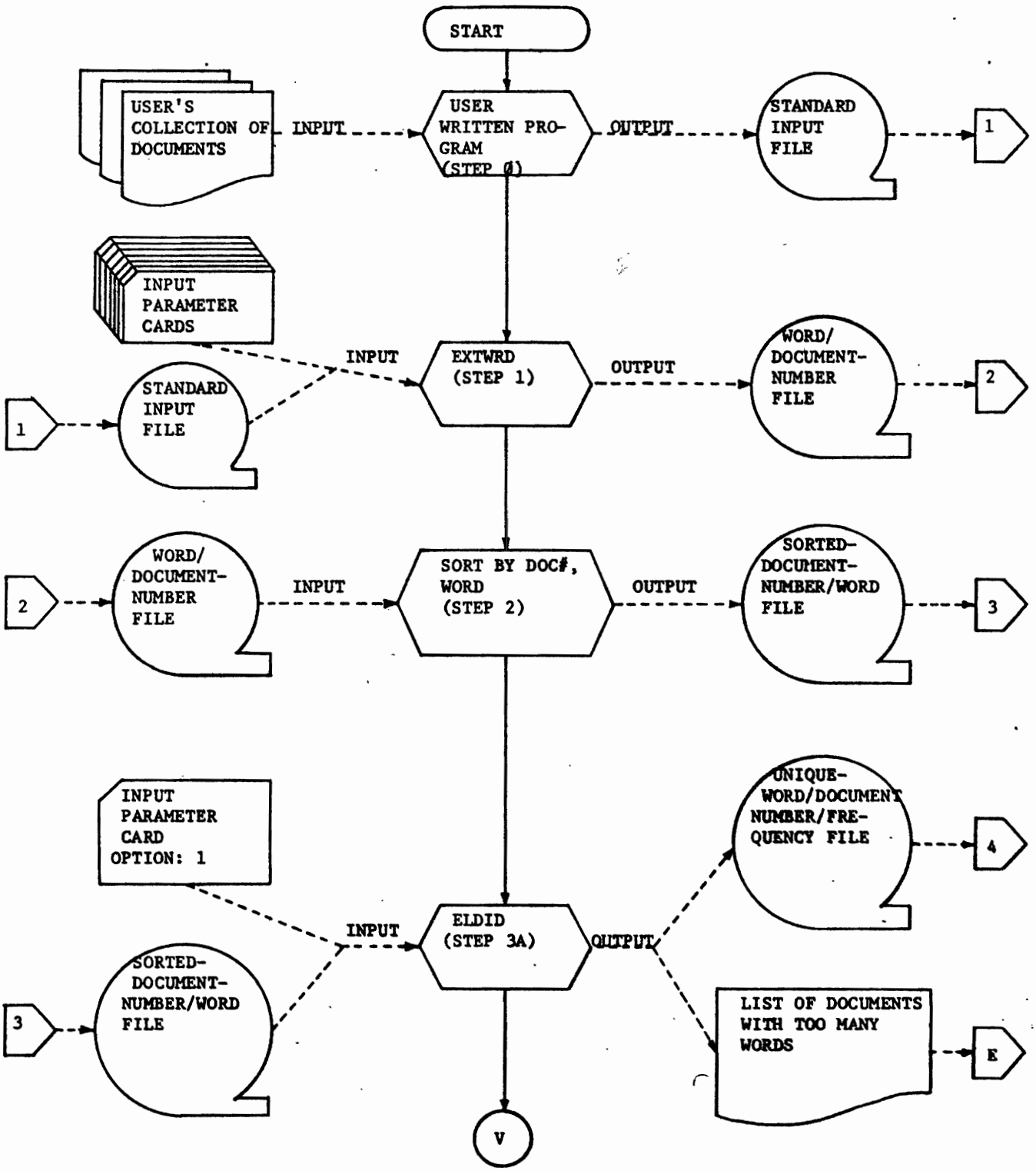


FIGURE 2.28
GENERAL FLOWCHART OF SEMI-AUTOMATIC INDEXING ROUTINES

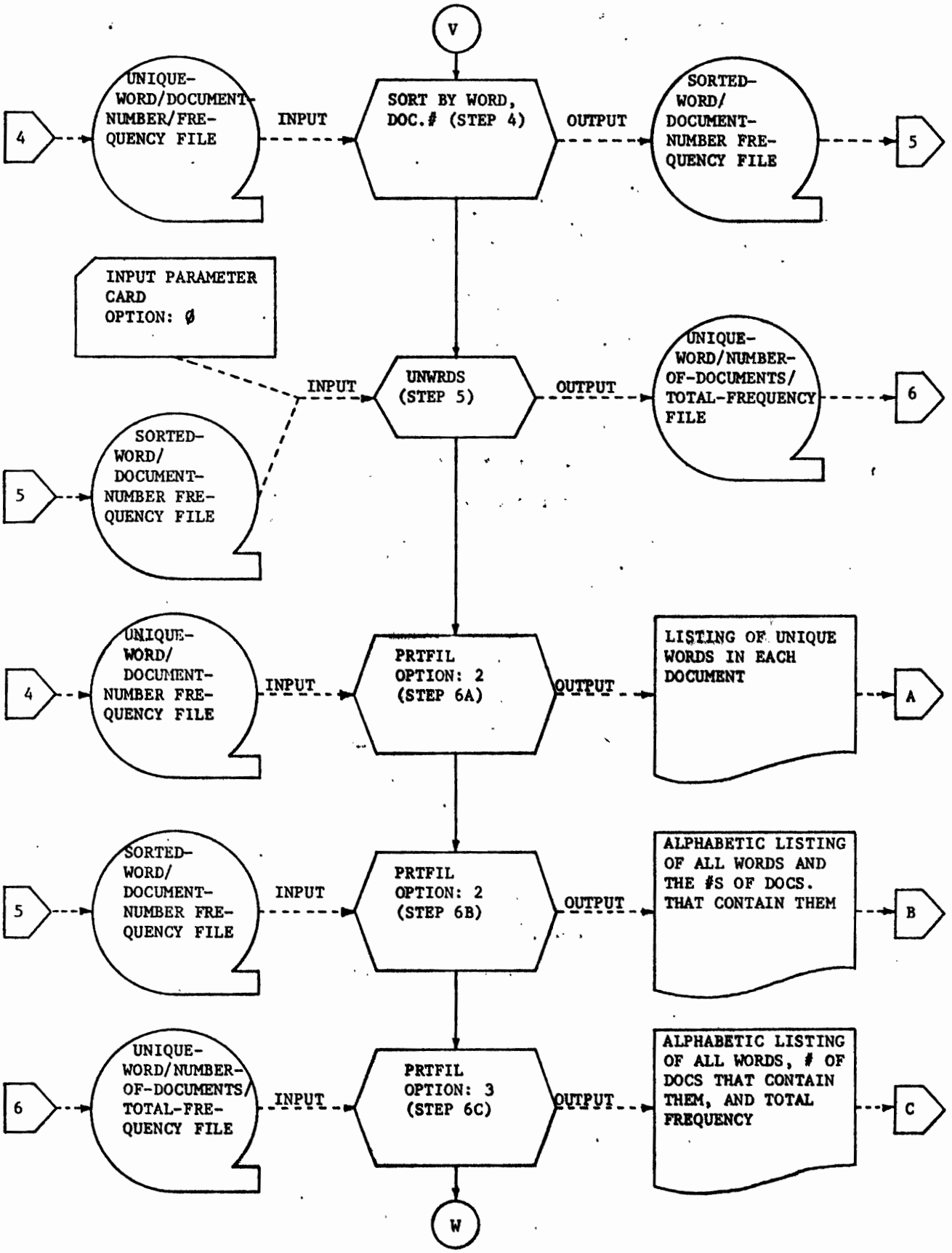


FIGURE 2.28
-CONTINUED-

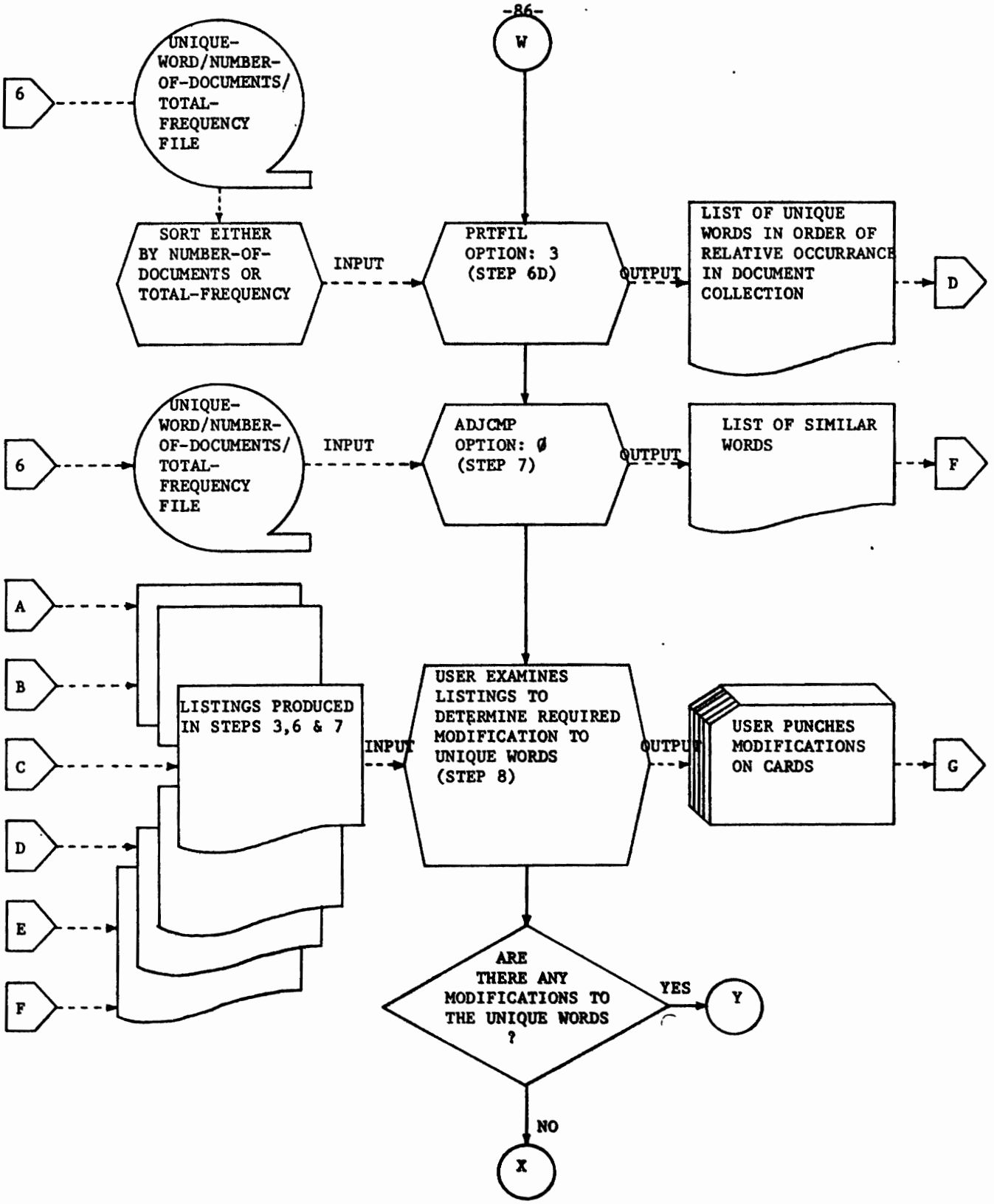


FIGURE 2.28
-CONTINUED-

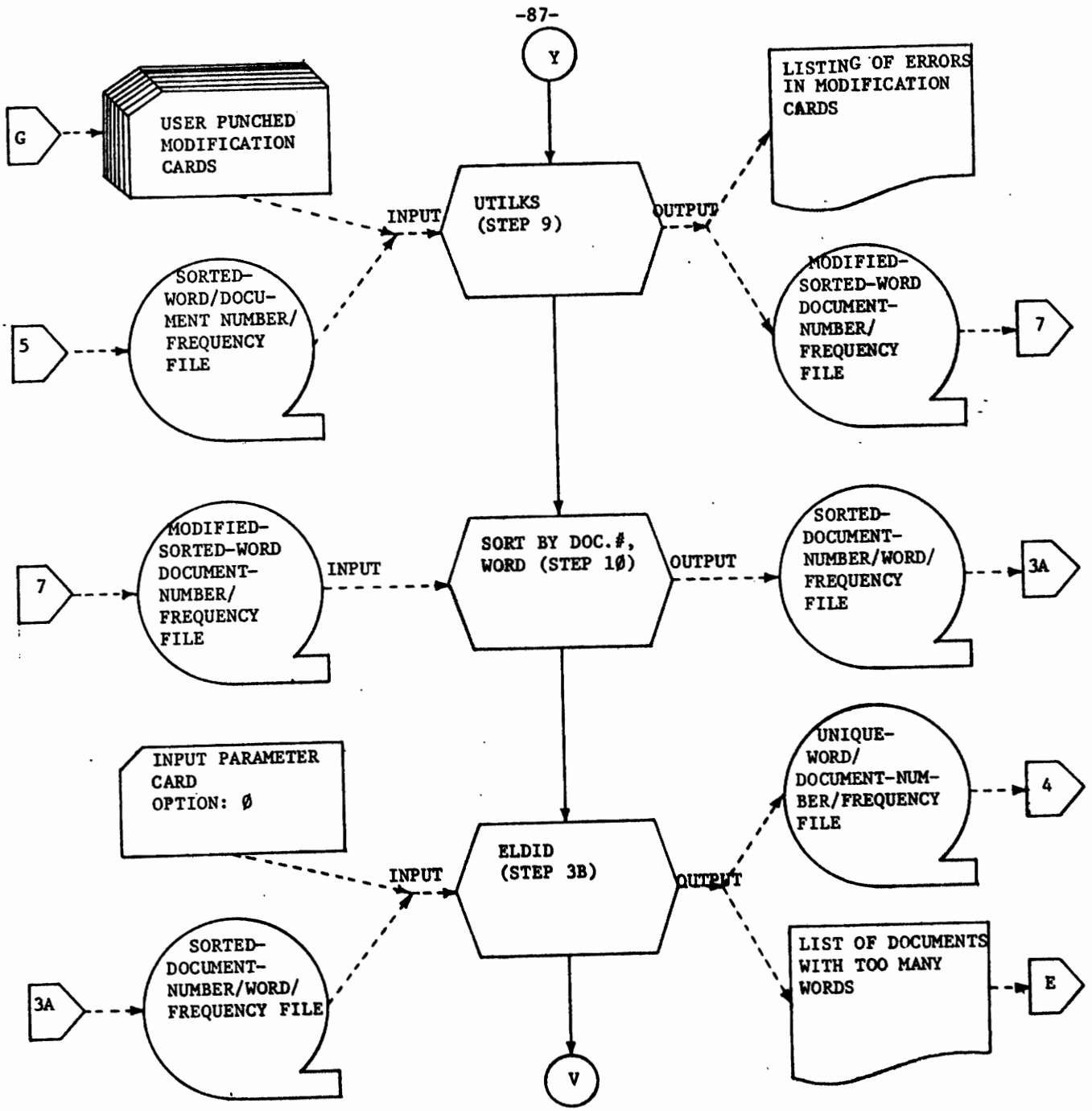


FIGURE 2.28
-CONTINUED-

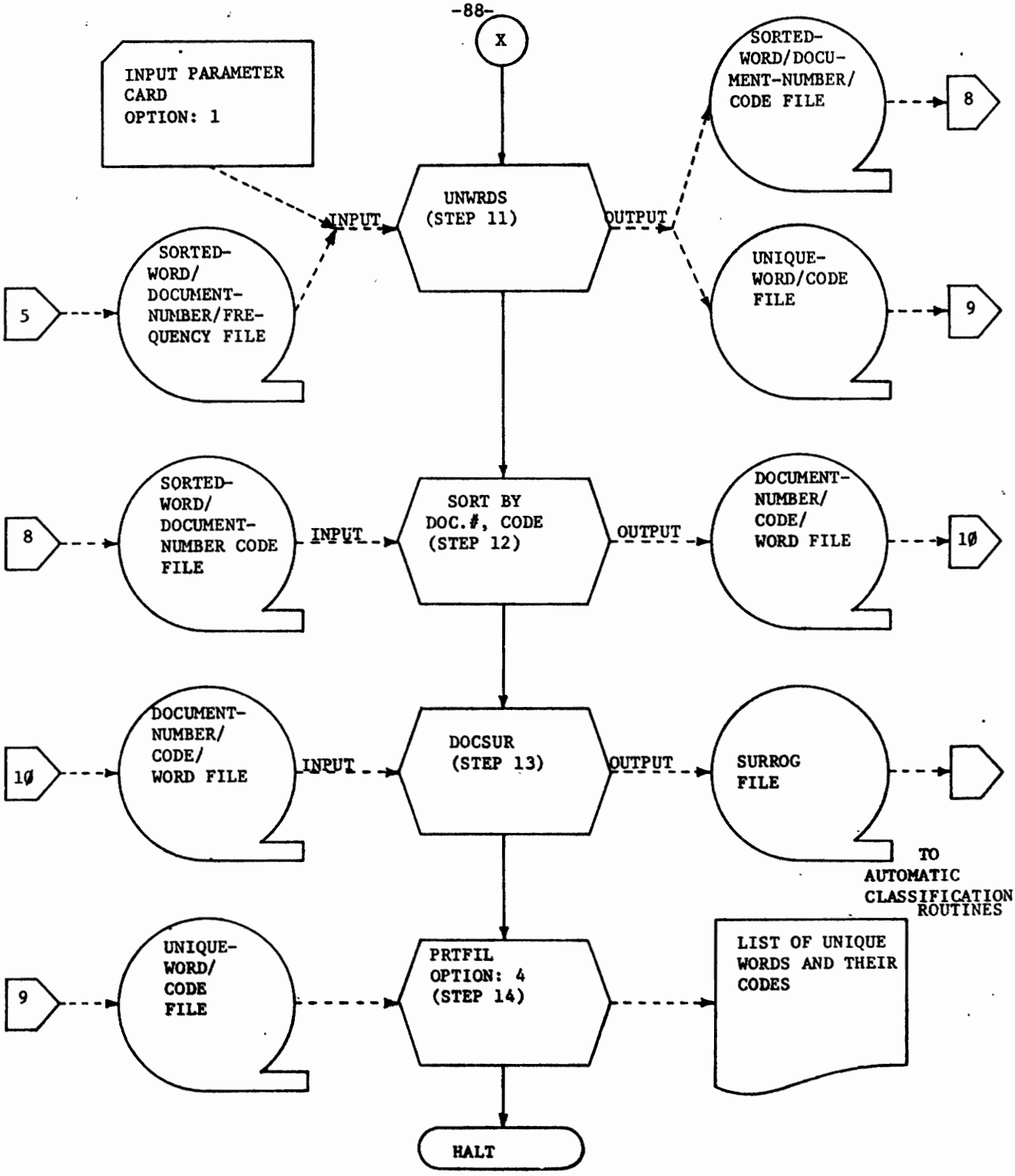


FIGURE 2.28
-CONTINUED-

Some of the steps given in this section require the user to sort the data on a file. Since generalized sort packages are standard software at most computer installations, a sort package is not included in the routines described in this paper. Most of the sort steps "alphabetize" the words on the input file. The "alphabetization" required by the Semi-Automatic Indexing Routine is unique in that the "blank" character must fall after the "z" (and not before the "a") in the collating sequence (see Figure 2.22). Each sort step refers to a figure where the sort field parameters, required as input to the sort, are described. These sort field parameters give the major through minor sort fields of the input record and the corresponding length of each field. The user must also provide his sort routines with the location of the fields within the input records to be sorted. The user may refer to the figures that describe the corresponding input file to each sort in order to determine the location of each field within the record. It should be noted that each character of the keyword in an input record is defined as a separate field whose length is two bytes and format is fixed point integer. By referring to each character of a keyword as a field, instead of the entire keyword as one large field, the sort routine will automatically "alphabetize" according to the required collating sequence.

When examining the examples of the semi-automatic indexing of the FBIS data base, the user should be aware of the following:

1) The maximum number of characters per word was set at 20. Each of the record lengths given in Figure 2.27 is computed using this value. (The keyword in each record occupies 40 bytes, i.e., twenty

characters at 2 bytes per character. The actual character is in the first of two bytes and a "blank" character is stored in the second. This is a constraint required by the FORTRAN language used to write the routines. The smallest addressable entity is a two byte field.)

2) The records on each file were written in unformatted (binary) mode. The Spectra 70 prefixes every unformatted record with a 4 byte control word; therefore, the actual length of a record can be computed by adding 4 to the record lengths given in Figure 2.27. The user must be aware of any such control bytes prefixed to unformatted records by the operating system of his computer. This is especially important when preparing the sort fields parameter card discussed in the previous paragraph. The length and format of each sort field is given to the user in the description of the sort fields parameter cards; however, the actual locations of the sort fields can only be computed from the file description after the user knows what, if any, control information is prefixed to his records. In the case of the Spectra, the first byte of data is actually the fifth byte in the data record due to the 4 byte control field prefixed to each unformatted record.

3) The files are created sequentially and may be either disk or tape volumnes. Tapes were used for the Spectra runs. The maximum blocksize that can be written by the Moore School's Spectra 70 is 4096 bytes.

4) Each file was referred to by the Spectra as FBISXX, where XX is the reference number of the file given in Figures 2.27 and 2.28.

5) The standard input (cards) and output (line printer) unit numbers were 5 and 6 respectively.

Each of the following steps corresponds to a processing box in the flowchart of Figure 2.28.

STEP 2 - Sort the Words in Each Document

Purpose: To arrange the words within each document in "alphabetical" order (Section 2.5).

Input: **WORD/DOCUMENT-NUMBER** File (FBISØ2)

Output: **SORTED-DOCUMENT-NUMBER/WORD** File (FBISØ3)

Example: The following is the deck setup and corresponding printout from the Spectra 70's standard sort package used to sort the input file by document-number, word: (The user is responsible for providing a sort routine, a standard utility at most computer installations, to sort the input file. The document-number should be the major field and each character of the word should be the minor fields; see Figure 2.7.)

```
/LOGON

/ERASE FBISØ2

/FILE FBISØ2,LINK=SORTIN,RECFORM=F,RECSIZE=48,BLKSIZE=4088,FCBTYPE=SAM, -
/VOLUME=FBISØ2,DEVICE=T9N,STATE=FOREIGN,OPEN=INPUT
(Definition of WORD/DOCUMENT-NUMBER FILE as input)

/FILE FBISØ3,LINK=SORTOUT,RECFORM=F,RECSIZE=48,BLKSIZE=4088,          -
/FCBTYPE=SAM,VOLUME=FBISØ3,DEVICE=T9N,OPEN=OUTPUT
(Definition of SORTED-DOCUMENT-NUMBER/WORD File as output)

/FILE PDISK,LINK=SORTWK,VOLUME=LANGØ1,DEVICE=D590,SPACE=(10000,1000)
(Definition of a private disk volume as work space for the sort. The
user can compute the amount of work space needed from the number of re-
cords on the input file and the length of each record.)

/EXEC SORT
(Command to run the sort routine)

SORT FIELDS=(45,4,A,5,2,A,7,2,A,9,2,A,11,2,A,13,2,A,15,2,A,17,2,A,19,2,A,
21,2,A,23,2,A,25,2,A,27,2,A,29,2,A,31,2,A,33,2,A,35,2,A,37,2,A,39,2,A,
41,2,A,43,2,A),FORMAT=FI
(This command tells the sort routine that the document-number is the
major sort field, and the individual characters of a word, from the 1st
to the 20th respectively, are the minor sort fields. All fields are Fixed
Point Integers and are sorted in Ascending order; see Figure 2.7.)
```

```
END
(The following messages were printed by the sort routine:)
S1Ø1     TSOS SORT/MERG DONE
S116     NUMBER OF OUTPUT RECORDS - 00000277177
S116     NUMBER OF INPUT RECORDS  - 00000277177
S116     NUMBER OF SORTED RECORDS - 00000277177
/LOGOFF
```

The "alphabetization" required by this step must correspond to the collating sequence in Fig. 2.22. The most notable exception to ordinary alphabetization is that the "blank" character sorts after the "z" instead of before the "a " The user must specify the sort fields according to Figure 2.7, thus assuring the correct "alphabetization," e.g., see SORT FIELDS command above.

STEP 3A - Run Program ELDID

Purpose: To eliminate the duplicate words within each document and assign each unique word a frequency-within-document value (section 2.6).

Input: (1) SORTED-DOCUMENT-NUMBER/WORD file (FBISØ3) - must be assigned FORTRAN unit number 21.

(2) Input Parameter Card. (Specify input option 1; see Figure 2.8) - read from FORTRAN unit number 5.

Output: (1) UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file (FBISØ4) - must be assigned FORTRAN unit number 22.

(2) Listing of the numbers of the documents that contain too many words. (The user specifies the maximum number of words allowed per document on the Input Parameter Card. The number of words per document must be $\leq$ 250.)

Example: (1) Deck Setup for Spectra 70:

```
/LOGON

/ERASE FBISØ3

/FILE FBISØ3,LINK=DSET21,RECFORM=F,RECSIZE=48,BLKSIZE=4088,                -
/FCBTYPE=BTAM,VOLUME=FBISØ3,DEVICE=T9N,STATE=FOREIGN,OPEN=INPUT
(Definition of SORTED-DOCUMENT-NUMBER/WORD file as input)

/FILE FBISØ4,LINK=DSET22,RECFORM=F,RECSIZE=50,BLKSIZE=4050,                -
/FCBTYPE=BTAM,VOLUME=FBISØ4,DEVICE=T9N,OPEN=OUTPUT
(Definition of UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file as output)

/EXEC LMELDID
(Command to run the load module corresponding to program ELDID.)

1ØØ2Ø1
(Input Parameter Card specifying (A) the maximum number of words per
document is 1ØØ, (B) the maximum length of a word in characters is 2Ø,
and (c) the input option is 1.)

/LOGOFF
```

(2) Sample output from program ELDID:

```
PROGRAM ELDID STARTED.

VALUE OF INPUT OPTION PARAMETER:  1

INPUT FILE ASSUMED TO BE:  SORTED-DOCUMENT-NUMBER/WORD

MAXIMUM KEYS PER DOCUMENT:  1ØØ

MAXIMUM LENGTH OF A WORD:  2Ø

        THE FOLLOWING DOCUMENTS HAVE MORE THAN 100 UNIQUE WORDS:

        DOCUMENT #     # OF WORDS

                1            266
                2            190
                3            248
                4            204
                8            210
                9            198
               10            188
               11            168
               12            118
               13            211
               15            123
               18            205
                .              .
                .              .
                .              .
             1644            152
             1645            143
             1648            128
             1649            118
             1650            106
             1651            113
             1652            105
             1655            181
             1657            168
             1658            180
             1662            139
             1663            155
             1664            256

NUMBER OF RECORDS WRITTEN ON UNIQUE-WORD/DOCUMENT/FREQUENCY FILE:  19Ø892

NUMBER OF DOCUMENTS WITH MORE THAN 1ØØ UNIQUE KEYS:  763

NUMBER OF DOCUMENTS IN COLLECTION:  1669

END OF PROCESSING FOR PROGRAM ELDID.
```

STEP 4 - Sort All Words

Purpose: To arrange the entire collection of words in "alphabetical" order (Section 2.7).

Input: UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file (FBISØ4).

Output: SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file (FBISØ5).

Example: The following is the deck setup and corresponding printout from the Spectra 70's standard sort package used to sort the input file by word, document-number: (The user is responsible for providing a sort routine, a standard utility at most computer installations, to sort the input file. Each character of the word, from the first to the last, must be the major sort fields and the document number the minor field; see Figure 2.12.)

```
/LOGON

/ERASE FBISØ4

/FILE FBISØ4,LINK=SORTIN,RECFORM=F,RECSIZE=5Ø,BLKSIZE=4Ø5Ø,FCBTYPE=SAM, -
/VOLUME=FBISØ4,DEVICE=T9N,STATE=FOREIGN,OPEN=INPUT
(Definition of UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file as input.)

/FILE FBISØ5,LINK=SORTOUT,RECFORM=F,RECSIZE=5Ø,BLKSIZE=4Ø5Ø,          -
/FCBTYPE=SAM,VOLUME=FBISØ5,DEVICE=T9N,OPEN=OUTPUT
(Definition of SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file as output.)

/FILE PDISK,LINK=SORTWK,VOLUME=LANGØ1,DEVICE=D59Ø,SPACE=(10000,1000)
(Definition of a private disk volume as work space for the sort. The
user can compute the amount of work space needed from the number of re-
cords on the input file and the length of each record.)

/EXEC SORT
(Command to run the sort routine.)

SORT FIELDS=(5,2,A,7,2,A,9,2,A,11,2,A,13,2,A,15,2,A,17,2,A,19,2,A,21,2,A,
23,2,A,25,2,A,27,2,A,29,2,A,31,2,A,33,2,A,35,2,A,37,2,A,39,2,A,41,2,A,
43,2,A,45,4,A),FORMAT=FI
(This command tells the sort routine that each character of the word,
from the 1st to the 20th, are the major sort fields, and the document
number is the minor field. All fields are Fixed Point Integers and are
sorted in Ascending order; see Figure 2.12.)
```

END

(The following messages were printed by the sort routine:)

S1Ø1 TSOS SORT/MERGE DONE

S116 NUMBER OF OUTPUT RECORDS - 00000190892

S116 NUMBER OF INPUT RECORDS - 00000190892

S116 NUMBER OF SORTED RECORDS - 00000190892

/LOGOFF

The "alphabetization" required by this step must correspond to the collating sequence in Fig. 2.22. The most notable exception to ordinary alphabetization is that the "blank" character sorts after the "z" instead of before the "a " The user must specify the sort fields according to Figure 2.12, thus assuring the correct "alphabetization," e.g., see SORT FIELDS command above.

STEP 5 - Run Program UNWRDS

Purpose: To eliminate the duplicate words within the entire document collection, and to assign each unique word two statistics: (1) the number of documents that contained the word and (2) the total frequency of occurrence (Section 2.8).

Input: (1) SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file (FBISØ5) - must be assigned to FORTRAN unit 21.

(2) Input Parameter Card. (Specify output option Ø; see Figure 2.13.)

Output: UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file (FBISØ6) - must be assigned to FORTRAN unit 22.

Example: (1) Deck setup for Spectra 70:

```
/LOGON

/ERASE FBISØ5

/FILE FBISØ5,LINK=DSET21,RECFORM=F,RECSIZE=5Ø,BLKSIZE=4Ø5Ø,FCBTYPE=BTAM,-
/VOLUME=FBISØ5,DEVICE=T9N,STATE=FOREIGN,OPEN=INPUT
(Definition of SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file as input.)

/FILE FBISØ6,LINK=DSET22,RECFORM=F,RECSIZE=52,BLKSIZE=4056,FCBTYPE=BTAM,-
/VOLUME=FBISØ6,DEVICE=T9N,OPEN=OUTPUT
(Definition of UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file as
output.)

/EXEC LMUNWRDS
(Command to run the load module corresponding to program UNWRDS.)

Ø2Ø
(Input Parameter Card specifying (A) the output option is Ø and (B) the
maximum number of characters per word is 2Ø.)

/LOGOFF
```

(2) Sample output from program UNWRDS:

```
PROGRAM UNWRDS STARTED.

OUTPUT OPTION PARAMETER:  Ø

MAXIMUM CHARACTERS PER WORD:  2Ø
```

UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY FILE BEING CREATED.

NUMBER OF RECORDS ON UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY FILE: 21473

PROCESSING FINISHED FOR PROGRAM UNWRDS.

STEP 6 - Run Utility Program PRTFIL

Purpose: This utility program should be used to list the contents of various files (Section 2.9).

Input: (1) Each time this program is run any one of the following files can be used as input - must be assigned FORTRAN unit 12.

A. UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY (FBISØ4)

B. SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY (FBISØ5)

C. UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY (FBISØ6)

D. The UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file can either be sorted, using the installation's standard sort package, by the number-of-documents or total-frequency fields and input to this run.

(2) Input Parameter Card - must be read from FORTRAN unit 5. This card (Figure 2.16) has two values: the input option and the number of records (words) to be printed. For each of the above input files, the following values should be used for the input option parameter:

A. Input Option: 2

B. Input Option: 2

C. Input Option: 3

D. Input Option: 3

Output: The following four listings are generated, depending upon which input file was used. Note: only one listing is generated per run. To get all four listings, program PRTFIL must be run four times, each time using a different file and the proper Input Parameter Card.

A. Listing of the unique words within each document. Each word is

printed along with its document number and a frequency that corresponds to the number of times the word occurred within the given document. All words in document one are listed first, then document two, etc.

B. Listing of all words in "alphabetical" order. Each word is printed along with its document number and frequency within document. If a word appears in several documents, then the word, document number, and frequency are listed for each document that contained the word.

C. List of unique words within the entire document collection. The words are listed in "alphabetical" order and each word is only printed once. Along with each word, two statistics are listed: (1) Number of documents that contained the word and (2) The total frequency of the word.

D. List of unique words in order of relative occurrence within the entire document collection. This listing is identical to the listing in C. above except that the words are either ordered by the number of documents that contained the word or the word's total frequency (depending upon which field the user chooses to sort the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file).

Example: (1) Deck setup from Spectra 70 for each possible input file:

A. /LOGON

```
/FILE FBIS04,LINK=DSET12,RECFORM=F,RECSIZE=50,BLKSIZE=4050,
/FCBTYPE=BTAM,DEVICE=T9N,OPEN=INPUT
(Definition of the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file as
input.)

/EXEC LMPRTFIL
(Command to run the load module corresponding to program PRTFIL.)

2 190892
(Data card specifying input option 2 and 190,892 records (words) to
```

be printed. The number of records on the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file may be obtained from the printout of program ELDID, i.e., Step 3.)

/LOGOFF

B. /LOGON

/FILE FBIS05,LINK=DSET12,RECFORM=F,RECSIZE=50,BLKSIZE=4050, -
/FCBTYPE=BTAM,DEVICE=T9N,OPEN=INPUT
(Definition of the SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file as input.)

/EXEC LMPRTFIL
(Command to run the load module corresponding to program PRTFIL.)

2 190892
(Data card specifying input option 2 and 190,892 records (words) to be printed. The number of records on the SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file is identical to the number of records on the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file. This value can also be obtained from the SORT printout in Step 4.)

C. /LOGON

/FILE FBIS06,LINK=DSET12,RECFORM=F,RECSIZE=52,BLKSIZE=4056, -
/FCBTYPE=BTAM,DEVICE=T9N,OPEN=INPUT
(Definition of UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file as input.)

/EXEC LMPRTFIL
(Command to run the load module corresponding to program PRTFIL.)

3 021473
(Data card specifying input option 3 and 21,473 records (words) to be printed. The number of records on the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file may be obtained from the printout of program UNWRDS, i.e., Step 5.)

/LOGOFF

D. Note; this deck setup shows the commands that first sort the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file by the NUMBER-OF-DOCUMENTS field and then runs program PRTFIL with the sorted file as input.

/LOGON

/ERASE FBIS06

```
/FILE FBISØ6,LINK=SORTIN,RECFORM=F,RECSIZE=52,BLKSIZE=4056,          -
/FCBTYPE=SAM,VOLUME=FBISØ6,DEVICE=T9N,OPEN=INPUT
(Definition of the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY
file as input to the sort routine.)

/FILE DUMY,LINK=SORTOUT,RECFORM=F,RECSIZE=52,BLKSIZE=4056,           -
/FCBTYPE=SAM,VOLUME=DUMY,DEVICE=T9N,OPEN=OUTPUT
(Definition of a scratch file named DUMY as output from the sort.)

/FILE PDISK,LINK=SORTWK,VOLUME=LANGØ1,DEVICE=D590,                   -
/SPACE=(10000,1000)
(Definition of a private disk volume as work space for the sort.
The user can compute the amount of work space needed from the number
of records on the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY
file and the length of each record.)

/EXEC SORT
(Command to run the sort routine.)

SORT FIELDS=(45,4,A),FORMAT=FI
(This command tells the sort routine that the number of documents
containing the word is the major sort field, this value begins with
the 45th byte of each record, is 4 bytes long, it is to be sorted in
Ascending sequence, and its format is Fixed Point Integer.)

END
(End of sort commands.)

/STEP

/ERASE DUMY

/FILE DUMY,LINK=DSET12,RECFORM=F,RECSIZE=52,BLKSIZE=4Ø56,            -
/FCBTYPE=BTAM,VOLUME=DUMY,DEVICE=T9N,OPEN=INPUT
(Definition of the file, DUMY, that was output from the sort and is
to be input to program PRTFIL.)

3 Ø21473
(Data card specifying input option 3 and 21,473 records (words) to
be printed. The number of records on the sorted file, DUMY, is
identical to the number of records on the UNIQUE-WORD/NUMBER-OF-
DOCUMENTS/TOTAL-FREQUENCY file.)

/LOGOFF
```

(2) Sample output from each of the four possible input files:

A.

KEYS	DOC. #	FREQ/DOC
.	.	.
.	.	.
.	.	.
UNITE	5	1
US	5	5
VISIT	5	1
WOMAN	5	1
AMBASSADOR	6	1
APRIL	6	1
BULGARIAN	6	2
.	.	
.	.	
.	.	
VIETNAM	6	3
VNA	6	1
WORKER	6	2
ADMINISTRATIVE	7	1
AGGRESSIVE	7	3
AIR	7	1
APRIL	7	1
BULGARIAN	7	4
CHINA	7	6
.	.	
.	.	
.	.	

This example shows the last few words in document 5, the words in document 6, and the first few words in document 7. Note that the word BULGARIAN occurred two times in document 6 and four times in document 7.

B.

KEYS	DOC. #	FREQ/DOC
AFRO-ASIAN	83	1
AFRO-ASIAN	485	1
AFRO-ASIAN	561	1
AFRO-ASIAN	833	2
AFRO-ASIAN	934	1
AFRO-ASIAN	1149	3
AFRO-ASIAN	1275	2
AFRO-SAIAN	1275	1
AFRO-SHIRAZI	1055	1
AFSUME	855	1
AFTERMATH	84	1
AFTERMATH	900	1
AFTERMATH	915	1
AFTERWARD	342	1
AFTERWARD	425	1
AFTERWARD	544	1
AFTERWARD	941	1
AGAINIST	58	1
AGAINIST	659	1
AGARTALA	632	1
AGARTALA	892	1
AGARTALA	1353	1
AGARTALA	1636	1
AGED	165	2
AGED	1272	1
AGENCYCORRESPONDENT	13	1
AGENCY	13	1
AGENCY	14	2
AGENCY	61	1
AGENCY	79	1
AGENCY	124	3

This example shows the words in "alphabetical" (note that AGENCY comes <u>after</u> AGENCYCORRESPONDENT because the "blank" character sorts after the "z") order. The word AFRO-ASIAN occurred in documents 83, 485, 561, 833, 934, 1149, and 1275. Its frequency of occurrenoe within each of the respective documents was 1, 1, 1, 2, 1, 3, and 2. (This information is summarized in the next example.)

C.

KEYS	# DOCS WITH KEY	TOTAL FREQ.
AFOREMENTIONE	6	6
AFORESAID	3	3
AFRAID	5	5
AFRICAN	23	39
AFRICA	34	46
AFRO-AMERICAN	5	22
AFRO-ASIAN	7	11
AFRO-SAIAN	1	1
AFRO-SHIRAZI	1	1
AFSUME	1	1
AFTERMATH	3	3

This example shows the words in "alphabetical" order. Note that the word AFRO-ASIAN occurred in seven documents with a total frequency of eleven. (These values were obtained by summarizing the information in the previous example.) "Similar" words such as AFRICAN and AFRICA will appear on the list of "similar" words which is output from program ADJCMP.

D.

KEYS	# DOCS WITH KEY	TOTAL FREQ.
RAHMANJ	1	1
RAKOV	1	1
PETROLEUM-EXPORT	1	1
PORK	1	2
NONCOMMUNIST	1	1
OVERCROWD	1	1
ONE-THIRD	1	1
RECREATIONAL	1	1
PLEKHANOV	1	3
QUEER	1	1
PRESSNOTE	1	3
REFINE	1	1
REFLEX	1	1
REFRIGERATOR	1	2
POLYMETALLIC	1	1
REFIT	1	1
DHANMAMDI	2	2
DEFLATE	2	2
DIGNITARY	2	2
DMITRY	2	2
EESE	2	2
DAZZLE	2	2
DEDUCE	2	2
EGON	2	2
DISAVOW	2	3
DAI	2	2
DESPISE	2	2
DMITRIY	2	3
DEGENERATE	2	2

This example shows some of the words that only appeared in one document (REFRIGERATOR with a total frequency of 2) and some of the words that appeared in only two documents (DIGNITARY with a total frequency of 2).

STEP 7 - Run Program ADJCMP

Purpose: To produce a listing of all "similar" words (Section 2.9).

Input: (1) There are two possible input files. The file that is chosen as input must be assigned FORTRAN unit 21.

A. UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY (FBISØ6)

B. SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY (FBISØ5)

(2) Input Parameter Card - must be read from FORTRAN unit number 5. This card (Figure 2.18) defines a "similar" word and contains an input option parameter. For each of the above input files, the following value should be used for the input option parameter:

A. Input option: Ø

B. Input option: 1

Output: Listing that contains groups of similar words.

A. If the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file is used as input (input option Ø specified) then each word will be printed with two statistics: (1) number of documents that contained the word and (2) the word's total frequency.

B. If the SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file is used as input (input option 1), then the list of "similar" words is produced without any statistics.

It is suggested that the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file be used as input since the corresponding statistics that are printed along with the "similar" word are helpful in determining which word(s) in a group of "similar" words are more important.

Example: The following example shows the deck setup and printout from program ADJCMP. The UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-

FREQUENCY file is used as input.

(1) Sample deck setup from Spectra 70:

```
/LOGON

/ERASE FBISØ6

/FILE FBISØ6,LINK=DSET21,RECFORM=F,RECSIZE=52,BLKSIZE=4056,          -
/FCBTYPE=BTAM,VOLUME=FBISØ6,DEVICE=T9N,STATE=FOREIGN,OPEN=INPUT
(Definition of the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file
as input.)

/EXEC LMADJCMP
(Command to run the load module that corresponds to program ADJCMP.)

Ø2ØØØ3ØØ2Ø
(Data card specifying: (1) the maximum number of characters per word is
2Ø. For adjacent words on the input file to be considered "similar"
(2) the first three characters of each word must match exactly and (3)
the maximum length of disagreement must be no greater than two. (Length
of disagreement is defined to be the number of characters between the
first pair of corresponding characters, in the words being tested for
"similarity", that do not match and the end of the longer word.)
Finally the input card specifies (4) that in input option is Ø.

/LOGOFF
```

(2) Sample output from program ADJCMP.

WORD	# DOCS WITH WORD	TOTAL FREQ.
AFRICAN	23	39
AFRICA	34	46
AGENDA	17	21
AGENT	39	84
AGRICULTURE	48	80
AGRICULTU	1	1
AHMAD	8	11
AHM	11	17
AIRLIFT	9	17
AIRLINE	11	18
AKAHATA	2	2
AKAHAT	1	1
ALAH	1	1
ALAN	1	1
ALARM	17	18
ALBANIAN	5	6
ALBANIA	5	7
ALEKSANDR	6	6
ALEKSANDUR	1	1
ALEKSEYEV	1	1
ALEKSEY	2	2
ALEXANDER	8	10
ALEXANDRA	1	2
ALEXANDRE	2	2
ALIA	2	2
ALIEN	11	12

This example shows groups of "similar" words, the number of documents containing each word, and the total frequency of each word. The user would want to CHANGE AFRICAN TO AFRICA with program UTILKS in order to normalize the set of unique words.

STEP 8 - Examining the Listings Produced in Steps 3, 6, and 7 (Section 2.9)

Purpose: To determine what modifications must be made to the unique words of the document collection in order to normalize these words.

Input: The user can examine any or all of the following six listings:

A. Listing of the numbers of the documents that contained more words than the user allows (Step 3). The user may want to change or delete some words in the documents that are too large.

B. Listing of the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file (Step 6). Since this listing contains the words grouped by document, the user may reference this list to find out exactly what words have been assigned to a particular document.

C. Listing of the SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY (Step 6). The user may reference this list to determine exactly what documents contained a particular word.

D. Listing of the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file (Step 6). This listing shows the unique words of the collection in "alphabetical" order.

E. Listing of the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file that has been sorted by either the number of documents or total frequency field (Step 6). This listing shows the unique words in order of relative occurrence in the document collection.

F. Listing of "similar" words (Step 7). This list can be used to determine misspellings or occurrences of several variations of the same root word.

Output: The user must either decide upon the modifications that must be made or that his collection of words are sufficiently free of

errors and the words are in a normal form (i.e., several variations of the same root word do not exist).

In the former case, a set of modification cards must be punched to be used as input to program UTILKS (Step 9).

In the latter case, the user may proceed directly to the steps that produce the document surrogates (Step 11).

The following six examples correspond to the six input listings mentioned above: (The decisions made in each example are taken from the sample listing given in Steps 3, 6, and 7.)

A. The total number of words in documents 1, 2, 3, . . . 1663, 1664 must be either reduced to $\leq$ 100 or the maximum number of words allowed per document increased. (Note that document 1 has 266 words. Since the maximum number of words cannot be greater than 250, document number 1 must be reduced by at least 16 words. Likewise document 1664 must be reduced by at least 6 words.)

B. Each of the documents that are too large may be examined individually. This can be a long process, especially if there are many documents with more words than the user allows. It is suggested that the user first make modifications to the entire collection, as opposed to particular document(s), then recycle getting another list of documents that are too large. Hopefully this new list will be small enough to either examine each document individually or redefine the maximum number of words per document to correspond to the largest document (must be $\leq$ 250).

C. The user may want to examine documents 83, 485, 561, 833, 934, 1149, and 1275 (using the listing in example B. above) to determine if AFRO-ASIAN is to be changed to either AFRO, ASIAN, or both. (The user cannot change AFRO-ASIAN to both AFRO and ASIAN

directly. He must CHANGE AFRO-ASIAN TO AFRO and ADD ASIAN to documents 83, 485, 561, 833, 934, 1149, 1275.)

The user may also want to examine document 1275 to determine if AFRO-SAIAN is misspelled. (The words AFRO-ASIAN and AFRO-SAIAN will only appear on the list of "similar" words if the user specifies a value of 5 or more for the length of disagreement parameter. This number is much too high since it will cause too many words to be displayed as "similar.")

D. The user may want to DELETE AFSUME as a nonsense word or CHANGE AFSUME TO ASSUME. The word AFOREMENTIONE occurred in six documents. The user may either delete it as having little importance or change it.

This listing is one of the most important. It gives the unique words in alphabetical order and may be used to locate candidates for deletion and changes.

E. Several words on this list (PORK, OVERCROWD, etc.) occur in only one document and only once in that document. The user may want to delete these as having little importance. If the user chooses to delete all words that occurred in one document with a frequency of one, then a computer program may be written to do this automatically.

The most frequently occurring words may also be examined individually. Since these words occur very often, they will have a profound influence on the classification. The user should carefully examine the most frequently occurring words deleting words with little importance.

F. Several words on this list should be changed since they are either misspelled or are variations of the same root word, e.g., CHANGE AFRICAN TO AFRICA, CHANGE AGRICULTU TO AGRICULTURE, and CHANGE ALBANIAN TO ALBANIA.

The list of "similar" words, along with the list of unique words in "alphabetical" order will be the most useful tools in determining the modifications to the unique words of the document collection.

If after examining the listings, the user decides there are no modifications to be made, then he is ready to create the document surrogates and should proceed to Step 11.

If the user decides that the unique words must be modified, then he should punch the corresponding modification cards (Figures 2.2Ø and 2.21) and recycle through the indexing process by proceeding to Step 9.

STEP 9 - Run Program UTILKS

Purpose: To start the re-cycling process by updating the words of the document collection according to the modifications decided upon in Step 8 (Section 2.9).

Input: (1) SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file (FBISØ5) - must be assigned FORTRAN unit number 21.

(2) Modification Cards - must be read from FORTRAN unit number 5 (Figures 2.2Ø and 2.21). These cards must be "alphabetized" on the first word in the card. The sequence in Figure 2.22 must be used when "alphabetizing" these words. (If the order of the modification cards corresponds to the order of the words on the UNIQUE-WORD/NUMBER-OF-DOCUMENTS/TOTAL-FREQUENCY file, then the user is assured of having the modification cards in the correct "alphabetical" order, since the words on this file are "alphabetized" according to the sequence in Figure 2.22.)

Output: (1) MODIFIED-SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file (FBISØ7) - must be assigned FORTRAN unit 22.

(2) Error Messages - any errors in the modification cards should be corrected. The user has the choice of either saving these corrections for the next re-cycle (the corrections must be merged with any modification cards punched for the new cycle) or performing a sort routine identical to Step 4 in order to re-sort the MODIFIED-SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file and then immediately re-running program UTILKS with the corrected modification cards.

Example: (1) Sample deck setup from SPECTRA 70:

```
/LOGON

/FILE FBISØ5,LINK=DSET21,RECFORM=F,RECSIZE=5Ø,BLKSIZE=4Ø5Ø,          -
/FCBTYPE=BTAM,DEVICE=T9N,OPEN=INPUT
(Definition of the SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file as input.)

/FILE FBISØ7,LINK=DSET22,RECFORM=F,RECSIZE=5Ø,BLKSIZE=4Ø5Ø,          -
/FCBTYPE=BTAM,VOLUME=FBISØ7,DEVICE=T9N,OPEN=OUTPUT
(Definition of the MODIFIED-SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file
as  output.)

/EXEC LMUTILKS
(Command to run the load module corresponding to program UTILKS.)

2Ø
(Data card giving the maximum number of characters per word.

DELETE ADVERSE
DELETE AFFECT
DELETE AFFORD
DELETE AFOREMENTIONE
DELETE AFRAID
DELETE AFTERNOON
DELETE AGAIN
DELETE AGERPR
DELETE AGGRAVATE
DELETE AGREE
DELETE AHEAD
DELETE AHM
DELETE AHTI
DELETE AIM
CHANGE ALLOCAT TO ALLOCATE
DELETE ALLOWE
DELETE ALLOW
DELETE ALLROUND
DELETE ALL
CHANGE EDUCAT TO EDUCATION
DELETE EFFECTIVENES
DELETE EFFECTIVE
DELETE EFFORT
DELETE EGC

(Modification cards, "alphabetized" by the first word on each card.
Note that the word ALL comes after ALLROUND since the "blank" character
sorts after the "z" in Figure 2.22.)

/LOGOFF
```

(2) Sample Error Messages:

```
****ERROR****   THE FOLLOWING COMMAND IS NOT RECOGNIZABLE AND IS -
                  IGNORED:
                (THE POINTER 'V' SHOWS WHERE PROCESSING STOPPED)
              ...V...
    'ERASE CPT TP CPU                                           '

****ERROR****   KEY TERM/DOCUMENT NUMBER CANNOT BE FOUND ON FILE, -
                  MODIFICATION CANNOT BE PERFORMED.
                KEY TERM: DEBTI          DOCUMENT NUMBER:     -1
                THE FOLLOWING COMMAND IS IGNORED:
    'CHANGE DEBTI TO DEBRIS                                     '

****ERROR****   COMMAND CARDS NOT PROPERLY SORTED BY FIRST KEY -
                  TERM FIELD.
                THE KEY TERM ON CURRENT COMMAND CARD: DECOARAT
                IS LESS THAN KEY TERM ON PREVIOUS COMMAND CARD: -
                  DECREE--SIGN
                THE FOLLOWING COMMAND IS IGNORED:
    'CHANGE DECOARAT TO DECORATE                                '
```

The above error messages show typical mistakes made in punching the modification cards. (1) The word TO was misspelled on the ERASE command. (2) The word to be modified was misspelled and could not be found on the input file. Since every occurrence of the word DEBTI was to be changed, the DOCUMENT NUMBER prints as -1. If the user would have indicated that DEBTI was to be changed only in specific documents (by punching a character in column 80 of the CHANGE card and including a Document-Number Card--see Fig. 2.21--that contained the documents to be modified) then the number of the document being processed when the error was discovered will be printed instead of "-1." (3) The modification cards were not properly "alphabetized." The card: CHANGE DECORAT TO DECORATE came after a card with the word DECREE--SIGN and was out of order since DECREE--SIGN should be before DECORAT.

The user should correct these errors for a later re-run of program UTILKS. The first two errors will cause new modification

cards to be punched:

ERASE CPT TO CPU

CHANGE DEBRI TO DEBRIS

STEP 1Ø - Sort by Document Number, Word

Purpose: To sort the words back into their respective documents.

This step is identical to Step 2 (Section 2.5).

Input: MODIFIED-SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file (FBISØ7).

Output: SORTED-DOCUMENT-NUMBER/WORD/FREQUENCY file (FBISØ3).

Example: Deck setup and printout from the Spectra 70's standard sort package used to sort the input file by document number, word: (the user is responsible for providing the sort routine to sort the input file. The document number must be the major field and characters of the word must be the minor fields; see Figure 2.7.)

```
/LOGON

/ERASE FBISØ7,LINK=SORTIN,RECFORM=F,RECSIZE=5Ø,BLKSIZE=4Ø5Ø,FCBTYPE=SAM,-
/VOLUME=FBISØ7,DEVICE=T9N,STATE=FOREIGN,OPEN=INPUT
(Definition of the MODIFIED-SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file
as input.)

/ERASE FBISO3

/FILE FBISØ3,LINK=SORTOUT,RECFORM=F,RECSIZE=5Ø,BLKSIZE=4Ø5Ø,FCBTYPE=SAM,-
/VOLUME=FBISØ3,DEVICE=T9N,OPEN=OUTPUT
(Definition of the SORTED-DOCUMENT-NUMBER/WORD/FREQUENCY file as output.)

/FILE PDISK,LINK=SORTWK,VOLUME=LANGØ1,DEVICE=D59Ø,SPACE=(10000,1000)
(Definition of a private volume as work space for the sort. The user can
compute the amount of work space needed from the number of records on the
SORTED-WORD/DOCUMENT-NUMBER FREQUENCY file and the length of each record.)

/EXEC SORT
(Command to run the sort routine.)

SORT FIELDS=(45,4,A,5,2,A,7,2,A,9,2,A,11,2,A,13,2,A,15,2,A,17,2,A,19,2,A,

21,2,A,23,2,A,25,2,A,27,2,A,29,2,A,31,2,A,33,2,A,35,2,A,37,2,A,39,2,A,41,

2,A,43,2,A),FORMAT=FI
(This command identifies the document number as the major sort field and
each character, from the 1st to the 20th, of the word as the minor fields;
see Figure 2.7.)

END
(End of sort commands.)
```

(The following messages were printed by the sort routine:)

```
S1Ø1   TSOS SORT/MERGE DONE
S116   NUMBER OF OUTPUT RECORDS - 00000100158
S116   NUMBER OF INPUT RECORDS - 00000100158
S116   NUMBER OF SORTED RECORDS - 00000100158
/LOGOFF
```

STEP 3B - Run Program ELDID

Purpose: To eliminate duplicate words within each document. This step is identical to Step 3A except that input option ∅ is specified in this case (Section 2.6).

Input: (1) SORTED-DOCUMENT-NUMBER/WORD/FREQUENCY file (FBIS∅3) - must be assigned to FORTRAN unit 21.

(2) Input Parameter Card (specify input option ∅; see Figure 2.8) - read from FORTRAN unit 5.

Output: (1) UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file (FBIS∅4) - must be assigned FORTRAN unit 22.

(2) Listing of the numbers of the documents that contained more words than the user allows.

Example: (1) Sample deck setup for Spectra 70:

```
/LOGON

/ERASE FBIS∅3

/FILE FBIS∅3,LINK=DSET21,RECFORM=F,RECSIZE=5∅,BLKSIZE=4∅5∅,          -
FCBTYPE=BTAM,VOLUME=FBIS∅3,DEVICE=T9N,STATE=FOREIGN,OPEN=INPUT
(Definition of the SORTED-DOCUMENT-NUMBER/WORD/FREQUENCY file as input.)

/FILE FBIS∅4,LINK=DSET22,RECFORM=F,RECSIZE=5∅,BLKSIZE=4∅5∅,          -
/FCBTYPE=BTAM,VOLUME=FBIS∅4,DEVICE=T9N,OPEN=OUTPUT
(Definition of the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY file as output.)

/EXEC LMELDID
(Command to run the load module corresponding to program ELDID.)

1∅∅2∅∅
(Input Parameter Card specifying (1) the maximum number of words per
document is 100, (2) the maximum number of characters per word is 2∅,
and (3) the input option is ∅.

/LOGOFF
```

(2) Sample Printout

The printout is similar to the example given in Step 3A. There were 96,999 records on the UNIQUE-WORD/DOCUMENT-NUMBER/FREQUENCY

file, and there were 297 documents with more than 100 words. (The largest document, number 156, had 202 words.)

The user should continue the re-cycle process by proceeding to Step 4. Recycling may be done as many times as necessary to produce a set of unique words that are relatively free from errors and contain mostly relevant keywords.

STEP 11 - Run Program UNWRDS

Purpose: To eliminate duplicate words within the entire document collection, and to assign a code number to each unique word (Section 2.8).

Input: (1) SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file (FBIS$\emptyset$5) - must be assigned FORTRAN unit number 21.

(2) Input Parameter Card (Specify output option 1- see Figure 2.13)

Output: (1) SORTED-WORD/DOCUMENT-NUMBER/CODE file (FBIS$\emptyset$8) - must be assigned FORTRAN unit number 23.

(2) UNIQUE-WORD/CODE file (FBIS$\emptyset$9) - must be assigned FORTRAN unit number 22.

Example: (1) Deck setup from Spectra 70:

```
/LOGON

/ERASE FBIS∅5

/FILE FBIS∅5, LINK=DSET21,RECFORM=F,RECSIZE=5∅,BLKSIZE=4050,FCBTYPE-
BTAM,-/VOLUME=FBIS∅5,DEVICE=T9N,STATE=FOREIGN,OPEN=INPUT
(Definition of SORTED-WORD/DOCUMENT-NUMBER/FREQUENCY file as input.)

/FILE FBIS∅8,LINK=DSET23,RECFORM=RECSIZE=5∅,BLKSIZE=4∅5∅,FCBTYPE=
BTAM,-/VOLUME=FBIS∅8,DEVICE=T9N,OPEN=OUTPUT
(Definition of the SORTED-WORD/DOCUMENT-NUMBER/CODE file as output.)

/FILE FBIS∅9,LINK=DSET22,RECFORM=F,RECSIZE=46,BLKSIZE=4094,FCBTYPE=
BTAM,-/VOLUME=FBIS∅9,DEVICE=T9N,OPEN=OUTPUT
(Definition of the UNIQUE-WORD/CODE file as output.)

/EXEC LMUNWRDS
(Command to run the load module corresponding to program UNWRDS.)

12∅
(Input Parameter Card specifying (1) that output option 1 is to be
taken and (2) the maximum number of characters per word is 2∅.)
```

(2) Sample output from program UNWRDS:

PROGRAM UNWRDS STARTED

OUTPUT OPTION PARAMETER: 1

MAXIMUM CHARACTERS PER WORD: 20

UNIQUE-WORD/CODE AND SORTED-WORD/DOC#/CODE FILES BEING CREATED

NUMBER OF RECORDS ON SORTED-WORD/DOC#/CODE FILE: 96999

PROCESSING FINISHED FOR PROGRAM UNWRDS.

STEP 12 - Sort by Document Number, Code

Purpose: To sort the words back into their respective documents in order to create a surrogate for each document (Section 2.1Ø).

Input: SORTED-WORD/DOCUMENT-NUMBER/CODE file (FBISØ8).

Output: DOCUMENT-NUMBER/CODE/WORD file (FBIS1Ø).

Example: The following is the deck setup and corresponding printout from the Spectra 70 standard sort package:

```
/LOGON

/ERASE FBISØ8

/FILE FBISØ8,LINK=SORTIN,RECFORM=F,RECSIZE=5Ø,BLKSIZE=4Ø5Ø,FCBTYPE=SAM,-
/VOLUME=FBISØ8,DEVICE=T9N,STATE=FOREIGN,OPEN=INPUT
(Definition of SORTED-WORD/DOCUMENT-NUMBER/CODE file as input.)

/FILE FBIS1Ø,LINK=SORTOUT,RECFORM=F,RECSIZE=5Ø,BLKSIZE=4Ø5Ø,          -
/FCBTYPE=SAM,VOLUME=FBIS1Ø,DEVICE=T9N,OPEN=OUTPUT
(Definition of DOCUMENT-NUMBER/CODE/WORD file as output.)

/FILE PDISK,LINK=SORTWK,VOLUME=LANGØ1,DEVICE=D59Ø,SPACE=(10000,1000)
(Definition of a private disk volume as work space for the sort.)

/EXEC SORT
(Command to run the sort routine.)

SORT FIELDS=(45,4,A,49,2,A),FORMAT=FI
(This command identifies the document number, which begins with the 45th
byte of each record and is 4 bytes long, as the major sort field and
the code number, which begins with the 49th byte and is 2 bytes long, as
the minor field; see Figure 2.23.)

END
(End of sort commands.)

(The following messages were printed by the sort routine.)

SlØ1   TSOS SORT/MERGE DONE

Sll6   NUMBER OF OUTPUT RECORDS - 00000096999

Sll6   NUMBER OF INPUT RECORDS - 00000096999

Sll6   NUMBER OF SORTED RECORDS - 00000096999

/LOGOFF
```

STEP 13 - Run Program DOCSUR

Purpose: To create a surrogate for each document. Each surrogate will consist of a record containing the document's number and the code numbers assigned to the words extracted from the document (Section 2.1Ø).

Input: (1) DOCUMENT-NUMBER/CODE/WORD file (FBIS1Ø) - must be assigned to FORTRAN unit 21.

(2) Input Parameter Card (see Figure 2.24) - read from FORTRAN unit 5.

Output: SURROG file - must be assigned to FORTRAN unit 22.

Example: (1) Sample deck setup from Spectra 70:

```
/LOGON

/ERASE FBIS1Ø

/FILE FBIS1Ø,LINK=DSET21,RECFORM=F,RECSIZE=5Ø,BLKSIZE=4Ø5Ø,FCBTYPE=BTAM,-
/VOLUME=FBIS1Ø,DEVICE=T9N,STATE=FOREIGN,OPEN=INPUT
(Definition of DOCUMENT-NUMBER/CODE/WORD file as input)

/FILE SURROG,LINK=DSET22,RECFORM=F,RECSIZE=52Ø,BLKSIZE=364Ø,          -
/FCBTYPE=BTAM,VOLUME=SURROG,DEVICE=T9N,OPEN=OUTPUT
(Definition of the document surrogate, SURROG, as output.)

/EXEC LMDOCSUR
(Command to run the load module corresponding to program DOCSUR.)

ØØ16692Ø22Ø
(This data card specifies (1) there are 1669 documents in the collection,
(2) the maximum number of words per document is 202. This value was ob-
tained from the listing output from Step 3B. The largest document is
number 156 with 202 words. If a value smaller than 202 is input, then
any document with more will have some words truncated from its surrogate.
These words, however, will be printed on an output listing. Finally
(3) the last field on the input card specifies 2Ø characters per word.)

/LOGOFF
```

(2) Sample printout from DOCSUR:

```
PROGRAM DOCSUR STARTED.
```

NUMBER OF DOCUMENTS IN COLLECTION (READ FROM INPUT CARD): 1669

MAXIMUM NUMBER OF KEY WORDS PER DOCUMENT: 2ø2

MAXIMUM NUMBER OF CHARACTERS PER WORD: 2ø

NUMBER OF RECORDS ON DOCUMENT SURROGATE FILE: 1669

PROCESSING FINISHED FOR PROGRAM DOCSUR.

File DOCSUR may now be input to the Automatic Classification Routines.

The number of documents value punched on the input card must be the actual number of documents in the collection. (This value is printed on the listing from Step 3B.) If this value does not match the number of records on the document surrogate file, a warning message is printed. The user should refer to Section 2.10.4 to determine the proper procedure to be taken if this message is printed.

STEP 14 - Run Program PRTFIL

Purpose: To produce an "alphabetical" list of the unique words in the document collection and their respective codes (Section 2.9).

Input: (1) UNIQUE-WORD/CODE file (FBISØ9) - must be assigned FORTRAN unit 12.

(2) Input Parameter Card (specify input option 4) - read from FORTRAN unit 5.

Output: "Alphabetical" listing of the unique words in the document collection and their respective codes.

Example: (1) Deck setup from Spectra 70:

```
/LOGON

/FILE FBISØ9,LINK=DSET12,RECFORM=F,RECSIZE=46,BLKSIZE=4094,
/FCBTYPE=BTAM,DEVICE=T9N,OPEN=INPUT
(Definition of the UNIQUE-WORD/CODE file as input.)

/EXEC LMPRTFIL
(Command to run the load module corresponding to program PRTFIL.)

4 ØØ5Ø53
(Data card specifying input option 4 and 5053 records (words)
to be printed.  The number of records on the UNIQUE-WORD/CODE
file may be obtained from the printout from program UNWRDS,
i.e., Step 11.)

/LOGOFF
```

(2) Sample printout from PRTFIL.

The reader should refer to the Appendix for a list of the unique words extracted from the FBIS data base.

CHAPTER 3

AUTOMATIC CLASSIFICATION

3.1 Introduction

The purpose of the group of routines described in this chapter, is to take the indexed document surrogate file, create a classified file complete with document text, document index terms, and category or cell to which the document belongs. Also, two dictionary files are created which will be used in making retrievals on the classified file. Table 3.1 is a list of the steps involved in the classification of the file. Figure 3.1 is a general system flow chart of the classification procedure indicating input and output from the programs at each step. Each of the programs involved will be described in this chapter. Five of the 9 programs are written in Fortran, and are machine independent. The other 4, are merely sorts and may be executed using the system sorting features of the computer performing the classification.

Step 1 - Program Name: CLASFY

Classification of the document surrogate file.

Step 2 - Program Name: Sort-One

Sort of the Final Keyword File, which is output from the classification. Sort is in descending node number sequence for input to the tree program. (Step 3)

Step 3 - Program Name: Tree

Creation of the classification Tree

Step 4 - Program Name: Sort-Two

Sort of the classification tree in ascending node number sequence.

Step 5 - Program Name - NOTOKY

Creation of Node to Key Dictionary, printing of Node to key Table, and creation of the Alphabetic Key-Node File.

Step 6 - Program Name - Sort-Three

Sorting of the Alphabetic key Node File in ascending alphabetic key sequence

Step 7 - Program Name: KYTOND

Creation of Key to Node Dictionary and Printing of Key-to-Node Table

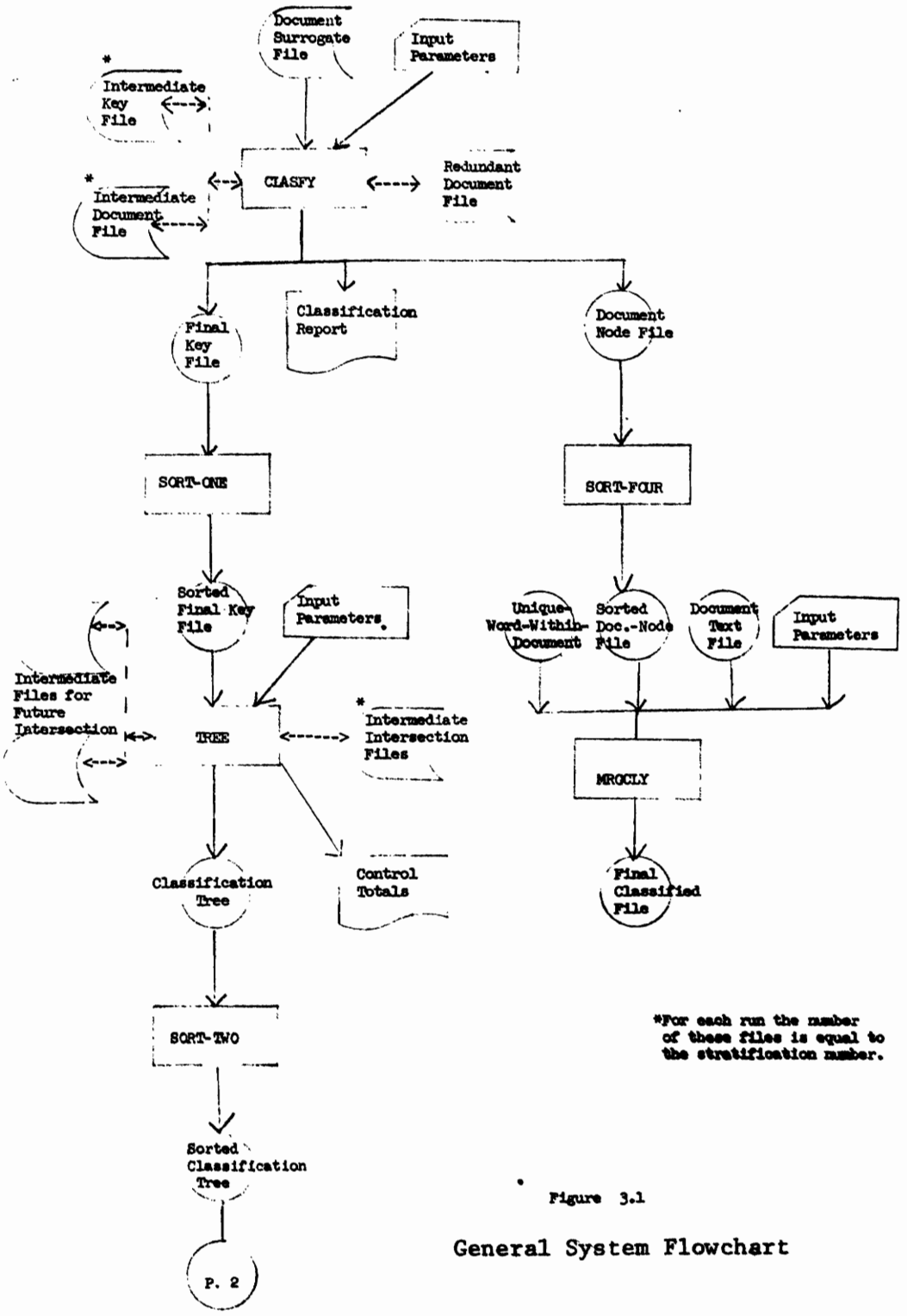
Step 8 - Program Name: Sort-Four

Sorting of Document Node File (which is output from CLASFY) for input to MRGCLY (Step 9)

Step 9 - Program Name: MRGCLY

Creation of final classified file containing text, alphabetic keys and cell number.

Table 3.1 - Steps in Classification



*For each run the number of these files is equal to the stratification number.

Figure 3.1

General System Flowchart

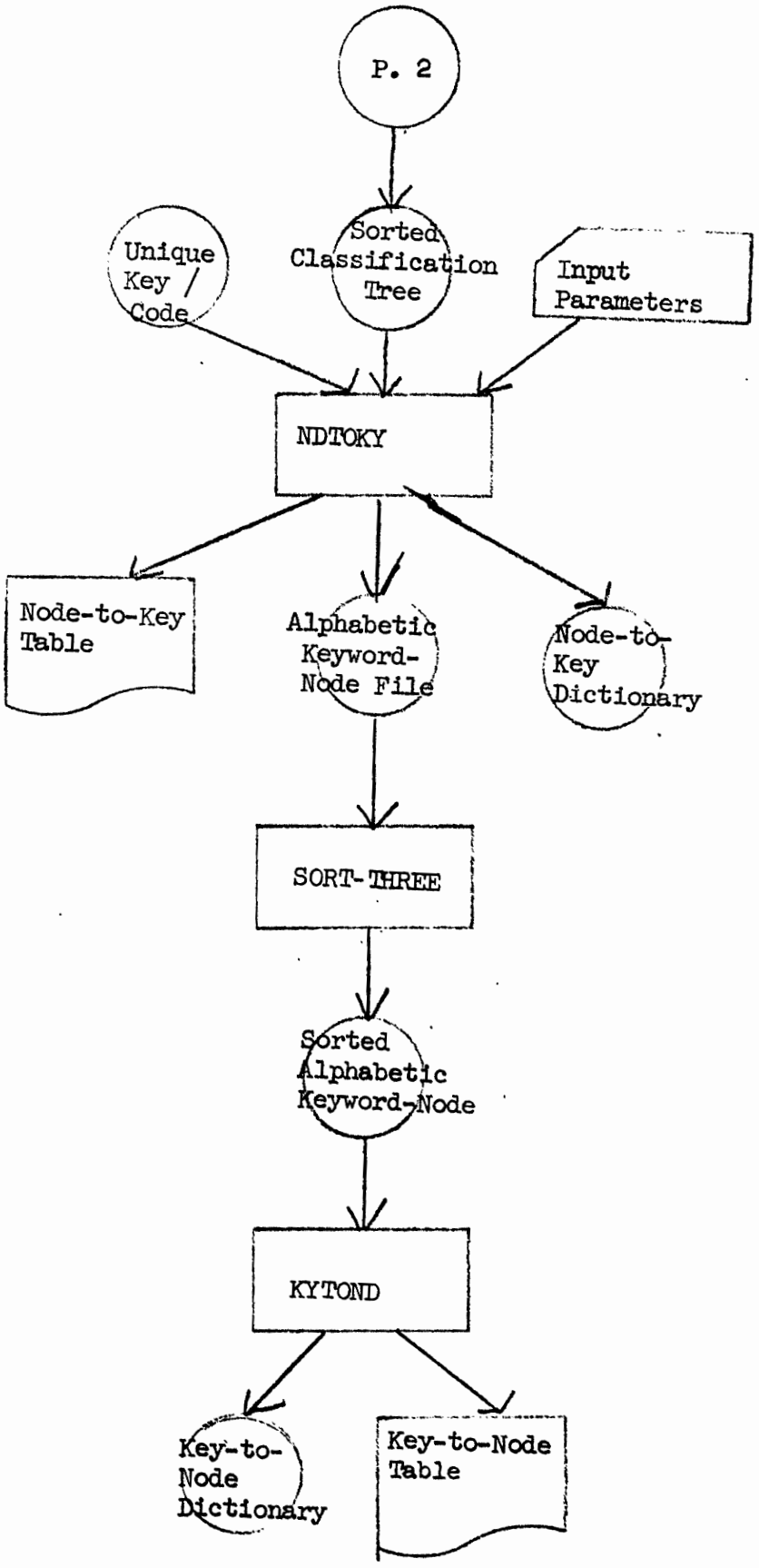


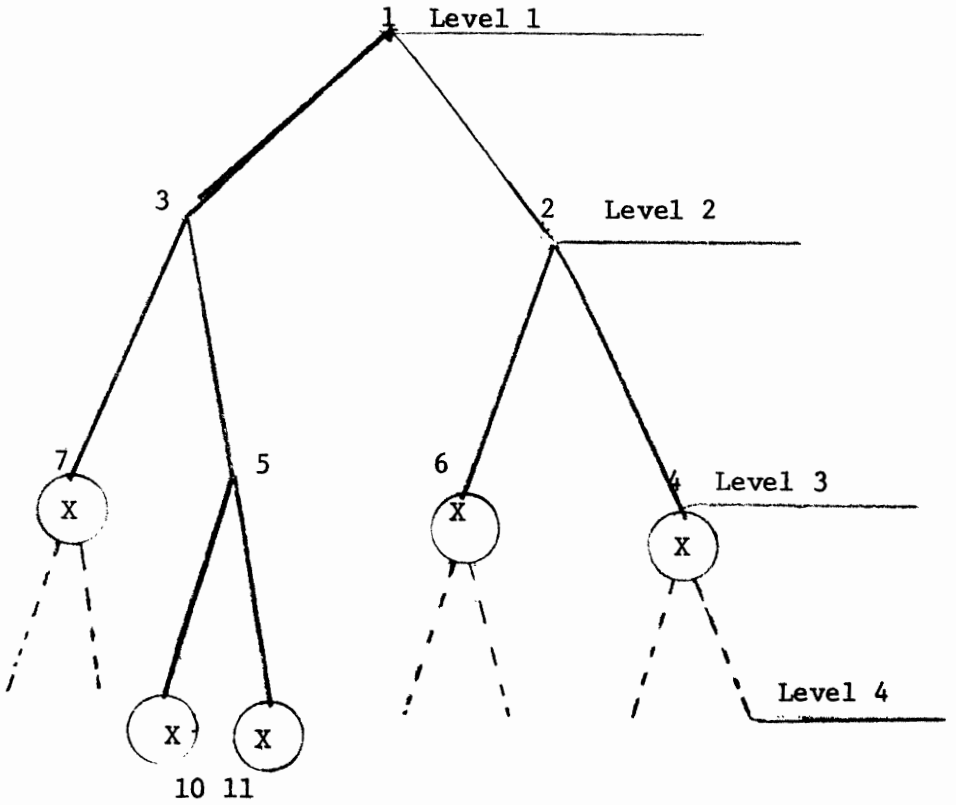
Figure 3.1, continued

3.2 CLASFY Description

This program takes the indexed document surrogate file, and classifies it using an algorithm called CLASFY, conceived by Dr. D. Lefkovitz [3]. The method of this algorithm is to start with the complete file and partition it into N groups. Then each of the N groups is subdivided into another set of N groups. This partitioning continues on each group until each group meets some maximum size criterion. That is, no group should contain more than the maximum number of documents allowed in each final group. Each group or node is treated independently of every other node and at any point in the classification, a node may be declared terminal with respect to size criteria. The final result is to obtain some number of mutually exclusive groups of documents.

In the process of creating the final groups, a tree is created (see fig. 3.2) with each group partition representing a node in the tree. This tree will be modified further on in the classification process (in Step 3) so it is here referred to as the intermediate classification tree. The final or terminal nodes (those which are not repartitioned) in the tree are referred to as cells.

In Figure 3.2a the tree of Figure 3.2 is illustrated. However, in this figure the canonical method of node numbering is introduced. This method of numbering allows one to determine the location of a node just from its number. For example, node'1.1.2'in the tree of Figure 3.2a can be found by starting at node one, then taking the first branch (indicated by the second '1' in 1.1.2) and then by taking the second branch (indicated by the '2' in 1.1.2). Since 1.1.2 has 3 digits in it, we also know that this node will be



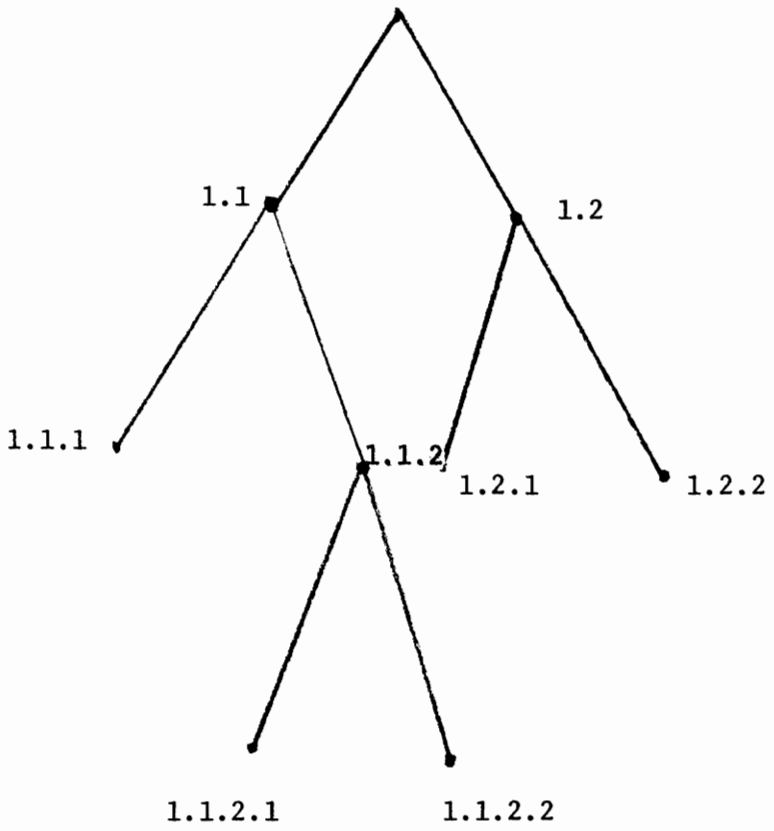
X terminal nodes

---- hypothetical partitions that might have occurred if the maximum cell size had been less than three

Node	No. of Documents	Terminal Node
1	12	
2	7	
3	5	
4	3	yes
5	4	
6	3	yes
7	2	yes
10	2	yes
11	2	yes

Fig. 3.2

Tree Formed in Clasification Process



Decimal Node No.	Canonical Node No
1	1
2	1.1
3	1.2
4	1.1.1
5	1.1.2
6	1.2.1
7	1.2.2
10	1.1.2.1
11	1.1.2.2

Fig. 3.2a

found in the third level of the tree. Figure 3.2a also shows the correspondence between the decimal numbers of Figure 3.2 and the canonical numbers. The canonical node numbering is the method to be used when discussing browsing and retrieval in the classified data base, and will be referred to in subsequent sections. Decimal node numbering will be used however, when convenient for descriptive purposes.

The tree created during the classification is not necessarily a balanced tree (i.e., one in which all terminal nodes are at the same level), since one node might meet the maximum cell criteria at one level while another node might need further repartitioning. In Figure 3.2 the tree created, in the process of classifying 14 documents is illustrated. The maximum cell size is three and this parameter governs the shape of the tree. This tree is not balanced. However, simulation of a balanced tree takes place in CLASFY in that numbering of the nodes will proceed as if the tree is balanced. In fig. 3.2 node 10 follows node 7 since node numbers 8 and 9 would appear as shown if terminal node 4 had had more than 3 documents and had been further subdivided. This balanced tree simulation is an important factor in the program that follow.

The final result of the classification is to have each document assigned to some terminal node or cell. This cell assignment is considered to represent the aposteriori category to which the document belongs. This end result is achieved in CLASFY in a

repetitive three pass process. Each partition is done in three passes of the file. When a partition is complete, the algorithm is repeated on each of the N groups independently. The selection of the number N, called the node stratification number is an input parameter to the CLASFY program and will be discussed in Section 3.2.7 which describes input to CLASFY.

3.2.1 Algorithm Description

The nodes of the tree in the classification are represented by the union of the keyword surrogates of the documents that are in the node. That is each node represents some set of β keywords from the complete vocabulary of keywords. The algorithm applied on a given node to form any N sibling nodes is based on three rules.

(1) The entire keyword vocabulary of the parent node must be divided so that every document description (i.e. all of its surrogates) will appear in at least one of the newly formed groups.

(2) The number of keyword terms in each group should be more or less even over the set of N groups.

(3) Each document description should appear in as few groups as possible.

Rule two is accomplished by setting an input parameter called the sensitivity factor E, to some value which prohibits the number of keywords in any group to differ from the number of keywords in any other group by more than this factor E. This parameter will also be discussed in section 3.2.7.

The following three sections will describe the 3 pass process which takes place in CLASFY.

3.2.1.1 PASS ONE

Each document description is read from the input file and added to one of the N groups as follows:

1) The group which contains the most keywords of the document is found and denoted as 'i'.
2) If there are two or more groups with an equal number of the document keywords, select the group that is the smallest.
3) If there are still two or more groups, after the execution of rule 2, then the group with the lowest group number is arbitrarily chosen.
4) At this point, before the keywords of a document are actually added to a group, the sensitivity factor E must be considered. Let the number of keys in group i be denote by n_i and the number of keys (of the document being processed) which are not in group i, be denoted by a_i.

The following criteria is tested:

$$n_i + a_i \leq n_j + a_j + E$$

That is, if the new group (after the current document keys are added) is no greater than the new size of any other group plus the sensitivity factor E, than the keywords are added. If not, the keywords are added to group j, where $(n_j + a_j)$ is a minimum for any j^{th} group.

At this point, when the whole file has been read, each document description is in at least one group. Now a test must be made to determine that at least two groups contain some keywords. For if only one group contains all the keywords, the classification is endless since each partition will result in one group containing all documents. This situation

could occur when the documents in the node being partitioned have few keywords per document and there is a large number of common keywords between documents. The most extreme case of this situation, would be a set of documents, each with an identical set of keywords. If the situation does arise, where only one group contains keywords, the node is artifically partitioned.

The artificial partition is done by first dividing the keywords in the one group into N equal groups and then beginning the process again at PASS one. PASS one would begin with each i^{th} group containing some keywords whereas PASS one in a normal partition would begin with all groups being empty.

3.2.1.2 PASS TWO

At the start of this pass, no documents have been assigned to any groups; only the keywords of the document have been assigned. At the end of the pass, those documents whose keywords appear in only one group, are assigned to the group, while those documents whose descriptions appear in more than one group are counted as redundancies and are written to an intermediate file for processing in PASS 3.

Documents whose keywords appear in only one group are assigned to the group by

1) Flagging the keywords of the document in that group. Flagging is accomplished by negating the integer key.

2) The document is written to one of N scratch files numbered coordinately with the key groups so that a document whose keywords appear in group i, is written to temporary scratch file i.

3.2.1.3 PASS THREE

If there are no redundancies, the partitioning is complete. If however, there are redundancies, PASS 3 is executed to minimize them.

1) Each document on the intermediate file is read, and the document is assigned to that group which has all the keywords of the document flagged. [The flagged keys are considered first, because they are already considered necessary to the group, by previous assignment of documents.] The first group with all of the document's keywords flagged is assigned and no other groups are checked.

2) If no group contains all flagged keywords of the document, then just consider those groups that contain all keywords of the document [from PASS 1 and PASS 2 we know that there must be at least two such groups]. Of these groups, select the group that has the most keywords of the document flagged. If there is more than one, choose the group with the lowest group number. The document is assigned to the chosen group by writing the record on the appropriate i^{th} scratch file, and the keys of the document not already flagged are flagged.

3.2.1.4 Partition Completion

Now all documents have been assigned to groups. The partition is complete. What remains, is to copy the N groups of documents from the temporary scratch files back to the main input file so that partitioning can be done again. Each of the N scratch files contains all the documents of a newly created node and represents a new independent group that may be further repartitioned. Each

group of documents is copied to the main input file, and in the process, the group is tested to see if it meets the minimum cell size. If it does

1) The first record of the group of documents is indicated to be the first record of a terminal cell.

2) The keywords associated with the group (or new terminal cell) are written on the final keyword file.

Note that only flagged keys are copied to the final keyword file since these are the only keywords that are considered essential to the cell.

3) The following information is written for each document on a file to be used in Step 9 of the classification system

a) document number

b) node number

c) number of keys in the document

The N scratch files are now free to be reused in the next partition.

3.2.2 CLASSIFICATION Example

The following is an example of a classification of 14 documents. Table 3.2 is a list of the 14 documents and their associated integer keys. Assume alphabetic keys have already been replaced by the integer keys. The fourteen documents will be partitioned by the CLASFY algorithm. For this example the stratification number will be equal to 3, the sensitivity factor will be equal to zero, and the maximum cell size will be equal to 3.

To begin, the complete file is designated as node 1, and node 1 has associated with it the complete keyword vocabulary consisting of keys 1 thru 15. Fig. 3.3 shows the results after each of the three

Document	Integer Coded Keywords
1	1 2
2	1 3 4
3	1 3 6
4	1 5 6
5	2 4 11
6	2 3
7	4 13 14
8	5 11
9	6 7 8
10	9 10
11	10 11
12	9 11
13	11 12
14	13 14 15

List of 14 Documents to be Classified

TABLE 3.2

PASS 1

Group	1	2
	1	1
	2	2
	4	3
Descriptions	5	4
	6	6
	7	9
	8	10
	9	13
	10	14
	11	15
	12	

(a)

PASS 2

Intermediate File

Document	Key
1	1, 2
10	9, 10

(b)

Documents Assigned

Group	1	2
	4	2
	5	3
	8	6
	9	7
	11	14
	12	
	13	
	1	
	10	

(d)

PASS 3

Keywords Assigned

Group	1	2
	1	1
	2	2
	4	3
	5	4
	6	6
	7	13
	8	14
	9	15
	10	
	11	
	12	

(c)

Fig. 3.3
First Partition for Classification Example

passes in the 1st partition of the file. After PASS 1 the keywords have been assigned to each group. No documents have as yet been assigned. After PASS 2, 12 of the fourteen documents have been assigned. Documents 1 and 10 are written on the redundant document file so that they may be assigned to a group in PASS 3. When PASS 3 is complete, all documents have been assigned to one of the 2 groups. These 2 groups now become known as node 2 and 3. Node 2 contains 9 documents and Node 3 contains 5 documents. After this initial partition, the partitioning continues on nodes 2 and 3 independently. That is the 9 documents in node 2 are divided into 2 more groups and the 5 documents in node 3 are also divided into 2 groups. This independent partitioning continues on each node until a node contains three documents or less. This procedure produces the tree structure illustrated in Fig. 3.4. (Note that this is not a balanced tree). There are 6 terminal nodes in the tree and the tree has 4 levels. Table 3.3 is a list of the final cell groups with their associated documents and keys.

Fig. 3.4 illustrates 2 important points about the classification.

1) When the classification is complete, only the terminal nodes have documents associated with them

2) Each node of the tree generated by CLASFY has associated with it the union of the keywords of the documents that belonged to the node before it was repartitioned (once repartitioning is done the documents no longer belong to the node that was partitioned). This tree, (with its associated keys) called the intermediate classification tree is modified in step 3 to form what is called the final

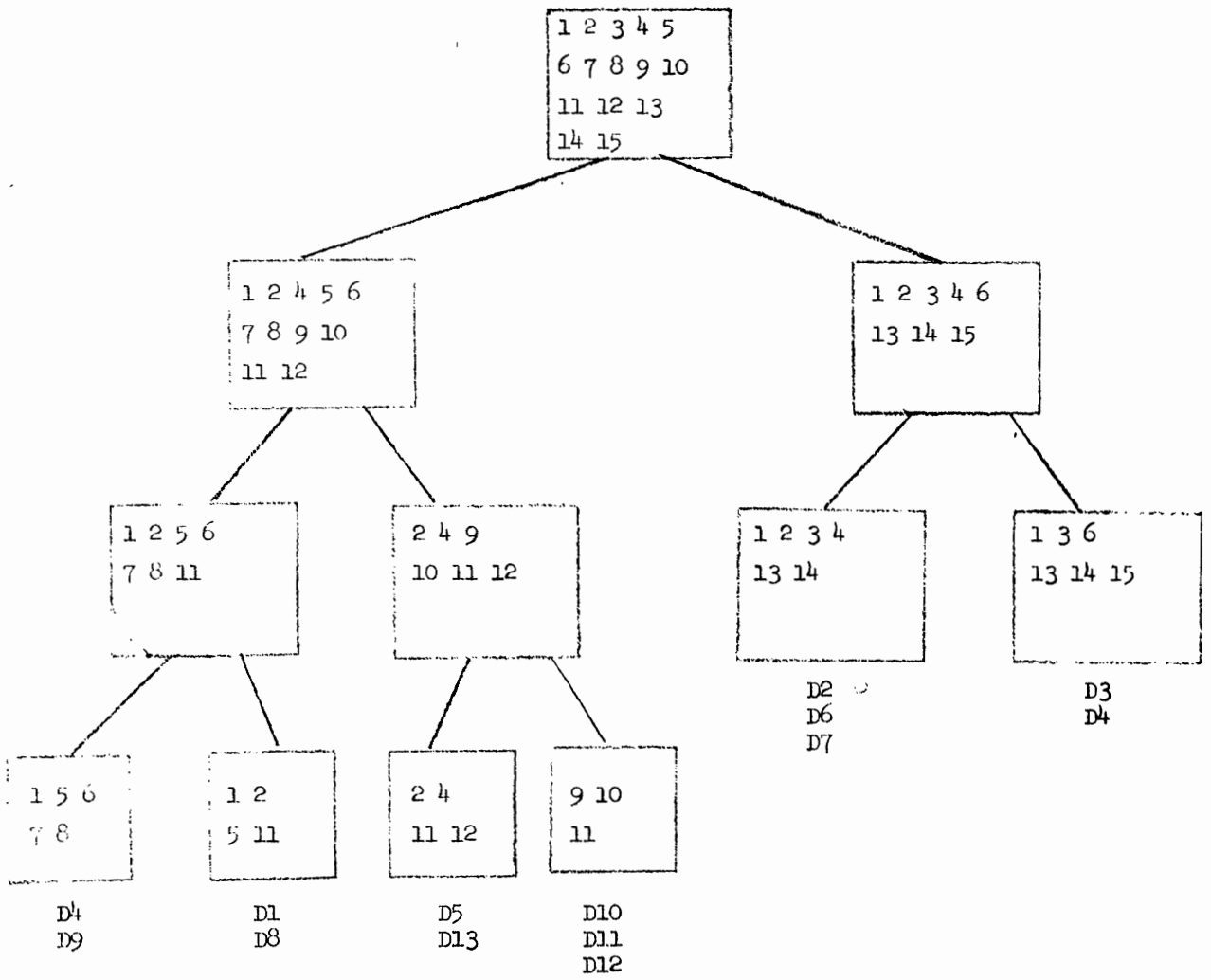


Fig. 3.4

Intermediate Classification Tree

Node Number	Doc. No.	Integer Keys
6	2	1 3 4
	6	2 3
	7	4 13 14
7	3	1 3 6
	14	13 14 15
8	4	1 5 6
	9	6 7 8
9	8	5 11
	11	10 11
10	5	2 4 11
	13	11 12
11	10	9 10
	11	10 11
	12	9 11

Final Cell Assignment For Example

TABLE 3.3

classification tree. The documents however are permanently assigned, at this point.

Fig. 3.5 is the actual output report produced as a result of the classification of the 14 documents.

3.2.3 Input Files

The document surrogate file is the input file to be classified and has the fields as indicated in Table 3.4. (Comments which follow are additions to those in Table 3.4). The file is in document number order (field 1) when it is initially partitioned. The number of keys in the document is indicated in field 2. Since this is a fixed length file, fields 7-256 are always present. However, only the number of fields indicated in field 2 will contain keys. The remaining key fields up to 250 will be blank.

Initially fields 3-6, pertain only to the first record in the file. That is, on initial input to CLASFY, the following appears in these fields:

Field 3 - contains the total number of records in the file

Field 4 - will be zero indicating this is not a terminal node and will be repartitioned. In this case, this is obvious since the file has not yet been partitioned at all.

Field 5 - contains the level at which this group or node became terminal. In this case it is zero; since the group (in this case the complete file) is not terminal, no level can as yet be indicated. This field is used for simulation of a balanced tree, as

```
CLASSIFICATION DATE          0
 TOTAL ITEMS                14        NODE       1
 PARTITIONS      2, E IS     0, CELL SIZE       3 ITEMS

 PHASE 1 CELL SWITCHES       2
 PHASE 2 REDUNDANCY          2
 GROUP    1       NO. OF KEYS    11     NO. OF ITEMS     9     NODE      2
 GROUP    2       NO. OF KEYS     8     NO. OF ITEMS     5     NODE      3
```


Figure 3.5

Output Report for Example

```
CLASSIFICATION DATE          0
 TOTAL ITEMS                 9        NODE       2
 PARTITIONS      2, E IS     0, CELL SIZE       3 ITEMS

 PHASE 1 CELL SWITCHES       0
 PHASE 2 REDUNDANCY          0
 GROUP    1        NO. OF KEYS     7     NO. OF ITEMS     4     NODE      4
 GROUP    2        NO. OF KEYS     6     NO. OF ITEMS     5     NODE      5
```

Figure 3.5, continued

```
CLASSIFICATION DATE        0

 TOTAL ITEMS               5       NODE      3

 PARTITIONS      2, E IS   0, CELL SIZE      3 ITEMS


 PHASE 1 CELL SWITCHES     2

 PHASE 2 REDUNDANCY        0

 GROUP    1      NO. OF KEYS     6     NO. OF ITEMS     3     NODE      6
 TERMINAL CELL

 GROUP    2      NO. OF KEYS     6     NO. OF ITEMS     2     NODE      7
 TERMINAL CELL
```

Figure 3.5, continued

```
CLASSIFICATION DATE          0

 TOTAL ITEMS                 4        NODE        4

 PARTITIONS      2, E IS     0, CELL SIZE        3 ITEMS


 PHASE 1 CELL SWITCHES       2

 PHASE 2 REDUNDANCY          0

 GROUP     1       NO. OF KEYS      5     NO. OF ITEMS      2     NODE       8
 TERMINAL CELL

 GROUP     2       NO. OF KEYS      4     NO. OF ITEMS      2     NODE       9
 TERMINAL CELL
```

Figure 3.5, continued

```
          CLASSIFICATION DATE          0

           TOTAL ITEMS                 5        NODE       5

           PARTITIONS      2, E IS     0, CELL SIZE       3 ITEMS


           PHASE 1 CELL SWITCHES       1

           PHASE 2 REDUNDANCY          0

           GROUP     1       NO. OF KEYS      4     NO. OF ITEMS      2     NODE      10
           TERMINAL CELL

           GROUP     2       NO. OF KEYS      3     NO. OF ITEMS      3     NODE      11
           TERMINAL CELL

NO. OF RECORDS ON KEY FILE =          12

NO.  OF LEVELS IN TREE =          4

NO. OF RECORDS ON DOCUMENT NODE FILE =        14

NO. OF NODES IN TREE =          11
```

Figure 3.5, continued

Field No.	Length in Bytes	Program Variable	Format	Description
1	4	NDN	Integer	Document Number
2	2	NDKY	Integer	No. of keys in this document
3	4	LEN	Integer	No. of records in this group
4	2	KCL	Integer	Terminal Cell Flag 0 - not a terminal cell 1 - indicates terminal cell
5	2	LEV	Integer	Indicates level in the tree at which the documents in the group became a terminal node
6	2	INODE	Integer	Terminal node no. on cell to which the group of records belong
7 . . . 256	2	KYSUR	Integer	Document keys--there can be from 1-250 keys with field #2 indicating the no. of keys in this document

TABLE 3.4 Document Surrogate File

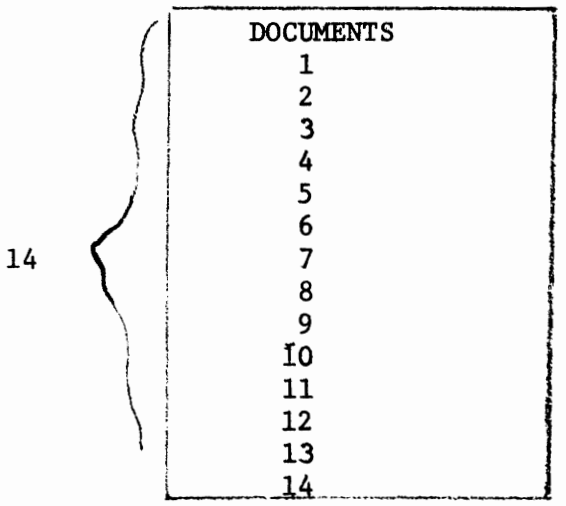
will be explained in Section 3.2.3.1

Field 6 - contains the node number to which this group of records belongs. In the initial case, all records belong to node 1 and this field is initially set to 1.

After partitioning has been done at least once, fields 3-6 pertain to the first record in each partitioned group. It is this first record of each group that contains the information (in Fields 3-6) necessary for further processing of the node. Field 3 contains the number of records in the node. It is field 3 which is tested to determine if the node meets the maximum cell size; and if it does, field 4 is set to one.

The document surrogate file is read sequentially at all times except when it is determined that a group is a terminal node. As was indicated in Section 3.2.1.2 during each partition, the documents are written on N temporary scratch files according to the group to which it is assigned. After each partition, the N groups that are formed are copied back from the scratch files to the document surrogate file.

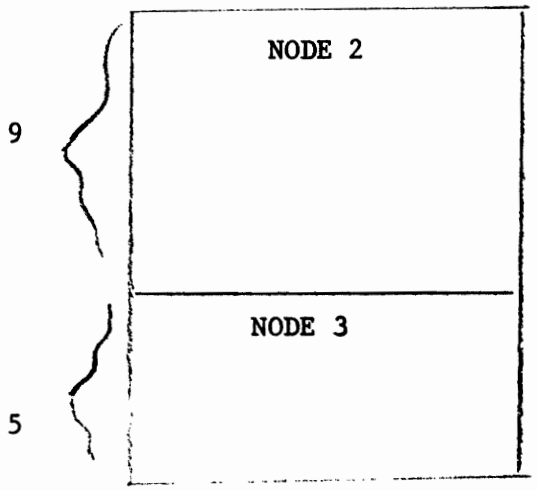
As an example, suppose initially the file looks as in Fig. 3.6a in accordance with the example given in Section 3.2.2. Record one would have the fields as indicated in the Figure. After the first partition the document surrogate file would look as in Fig. 3.6b with each rectangle indicating a node or group and containing the records assigned to it in PASS 2 and PASS 3. Node 2 contains 9 records and Node 3 contains 5 records. The first record in node 2 and the 1^{st} record in node 3 would contain the values as shown in



For Record 1 in the File

Field	Value
3	14
4	0
5	-
6	-

Fig. 3.6a



For Record 1 of Node 2

Field	Value
3	9
4	0
5	-
6	-

For Record 1 of Node 3

Field	Value
3	5
4	0
5	-
6	-

Fig. 3.6b

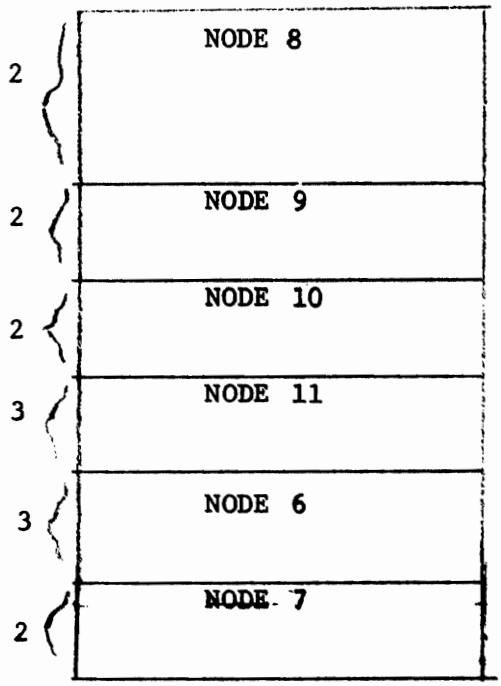


Fig. 3.6c

Fig. 3.6

Example Of Input File

the figure. The final classified file in the example would look as in Fig. 3.6c and the first record in each node would contain a 1 in field 4 indicating that all nodes are terminal and no repartitioning is to be done.

3.2.3.1 Balanced Tree Simulation

To illustrate the use in CLASFY of Fields 4 and 5 (in the document surrogate file) for maintaining a balanced tree numbering scheme, the example in Section 3.2.2 will be altered. Suppose we again start with 14 documents and use a stratification number of 2, and a maximum cell size of 3. This time, suppose after the second level partitioning is complete, the file contains that as shown in Fig. 3.7a. Since the maximum cell criteria is 3, only node six will need further repartitioning. Therefore the first records of nodes 4, 5, and 7 will contain a 1 in field 4 indicating a terminal node and a 3 in field 5 indicating that this terminal node is at level three of the tree. (fig. 3.7b) Upon completion of level 2 partitioning, CLASFY begins again at the beginning of the document surrogate file, to search for nodes that need further repartitioning. The search would proceed as follows:

1) The 1^{st} record of node 4 would be read. The terminal cell indicator would be checked. When it was found to be equal to 1, the entire node would be skipped and the next record accessed would be the first record of node 5. Also, to maintain a balanced tree, the balanced tree node numbering must be continued. This is done by skipping those numbers that would have been assigned if node 4 had actually been further subdivided. The formula used to

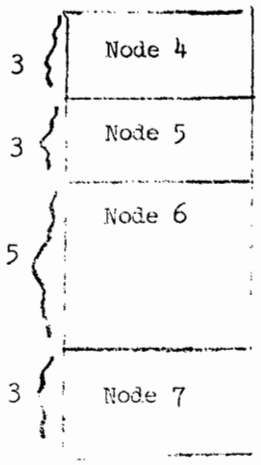


Fig. 3.7a

Document Surrogate File
After 2nd Level Partition

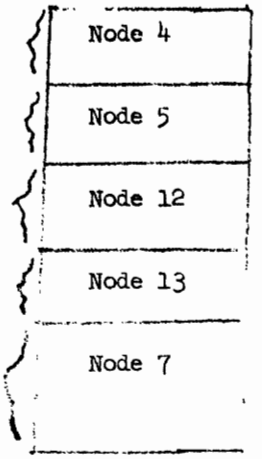


Fig. 3.7c

Document Surrogate File
at End of Classification

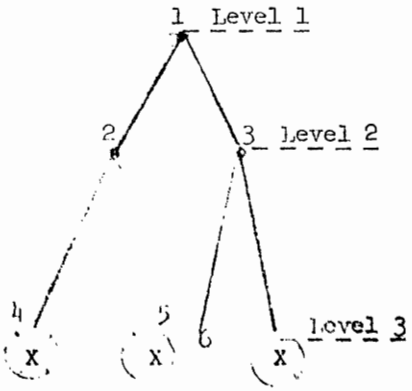


Fig. 3.7b

Tree Formed After
2nd Level Partition

(X) Terminal Nodes

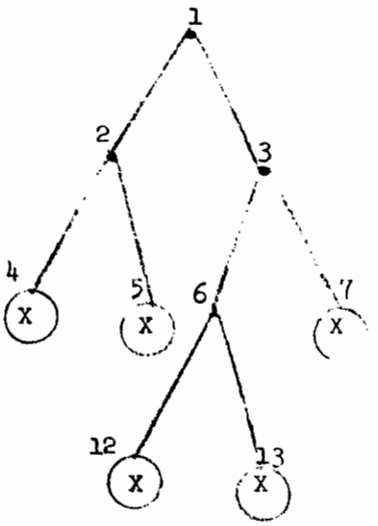


Fig. 3.7d

Tree Formed at End of Classification

calculate the number of integer nodes to be skipped is:

$$SKIP = N^{x-y}$$

where

N = The Stratification Number

y = The level at which the node became a terminal cell (field 5)

x = The next level of the tree to be created

In the example given above, for node 4,

$$SKIP = 2^{(4-3)} = 2$$

therefore 2 node numbers would be skipped.

2) The 1st record of node 5 would then be accessed and just as in node 4, since node 5 is terminal, all the records in the node would be skipped and the number of integer nodes to be skipped would be calculated.

3) The first record of node six would be accessed and found to need further repartitioning; the partioning would be done.

4) The 1st record in node 7 would be accessed and subsequently skipped.

Assuming all nodes are now terminal, classification is complete and the classified file would appear as shown in Fig. 3.7c and the classification tree as in Fig. 3.7d.

3.2.4 Output Files

3.2.4.1 Final Keyword File

The final keyword file is output from CLASFY and contains all the keys associated with each of the terminal nodes of the classification tree. Table 3.5 contains the list of fields in the records in this file. Each terminal node has associated with it, a header record indicating the node number and number of keys in the node,

Field No.	Length in Bytes	Program Variable	Format	Description
Header Record				
1	2	INODE	Integer	Terminal Node Number
2	2	ISEQ	Integer	Sequence No.
3	2	INKY	Integer	No. of keys in this terminal node
Keyword Record				
1	2	INODE	Integer	Terminal Node Number
2	2	ISEQ	Integer	Sequence No.
3	2	JGRP	Integer	Integer Key
. . . maximum allowed				

TABLE 3.5

Final Keyword File

and as many keyword records as it takes to hold all the keywords in the node. The maximum number of keywords contained in each keyword record is an input parameter and will be described in Section 3.2.7.

The sequence number in both the header record and the keyword record is used to keep a count of the number of records associated with each node.

The final keyword file contains the information that will be used in the next steps to create the dictionaries associated with the classification.

3.2.4.2 Document-Node File

The document node file has the fields as shown in Table 3.6. The purpose of this file is to aid in the creation of the final classified file, which will contain complete document text, document keys, and node or cell assignment. This file is used as a dictionary for that purpose. Each classified document number will appear in this file along with its node number and number of keys in the document. When the final classified file is created, the document node file will be used as a dictionary to look up the correct node number and number of keys associated with the document.

3.2.5 Intermediate Files

There are a number of intermediate or scratch files used in CLASFY. These files are used within CLASFY and serve no use when CLASFY is complete. They may be erased at the conclusion of the program.

3.2.5.1 Redundant Document File

The redundant document file contains document records of those

Field No.	Length in Bytes	Program Variable	Format	Description
1	4	NDN	Integer	Document Number
2	2	JCURND	Integer	Terminal Node Number
3	2	NDKY	Integer	Number of keys in the document

TABLE 3.6

Document - Node File

documents that have not been assigned to a group during PASS 2 of CLASFY. At the end of PASS 2 this file is rewound and used as input to PASS 3.

3.2.5.2 Intermediate Document Files

The number of intermediate document files is equal to N, the stratification number for the classification. These files are used for storing the document records assigned to a group in a partition and must therefore be equal in number to the number of groups that can be formed. Records are stored on these files during PASS 2 and PASS 3 and when partitioning is complete upon a node, the records are copied back to the main input file so that these files may be reused in the next partition.

3.2.5.3 Intermediate Key Files

The number of document key files is also equal to N, the stratification number for the classification. These files are used for storing the keys associated with the documents in a particular group of a partition. As stated in Section 3.2.1.1, it is in PASS 1 that these key groups are formed. During PASS 2 and PASS 3 these key files are used for assignment of the documents to groups. Once a partition is complete, the intermediate key files are no longer needed unless it is determined that a group is a terminal cell. In that case, the key group file associated with the terminal cell is copied to the final key word file. When all terminal key groups have been copied, all key group files are ready to be reused for the next partition.

3.2.6 Classification Output Record

For each partition in the CLASFY program, a summary report is printed. An example of such a report is shown in Figure 3.8. The circled numbers indicate each of the elements in the report.

(1) date of the run

(2) total number of documents in the current node being partitioned

(3) node number being partitioned

(4) (5) and (6) - input data, and will be the same for each node in the classification

(4) stratification number

(5) sensitivity factor which determines the maximum difference in size between key groups

(6) maximum number of documents that can appear in any terminal cell

(7) this field is generated in PASS 1. It occurs as a result of the sensitivity factor test as described in Section 3.2.1.1. That is, if when adding keys of a document to a group i, it is determined that the following is not true

$$N_i + a_i \leq n_j + a_j + E$$

then the group j becomes the new group i. That is, group j becomes the group to which the current document keys are added and phase 1 cell switch counter is incremented by 1. $N_j + a_j$ must be a minimum over all j, and there can therefore be more than 1 cell switch for each complete sensitivity factor test.

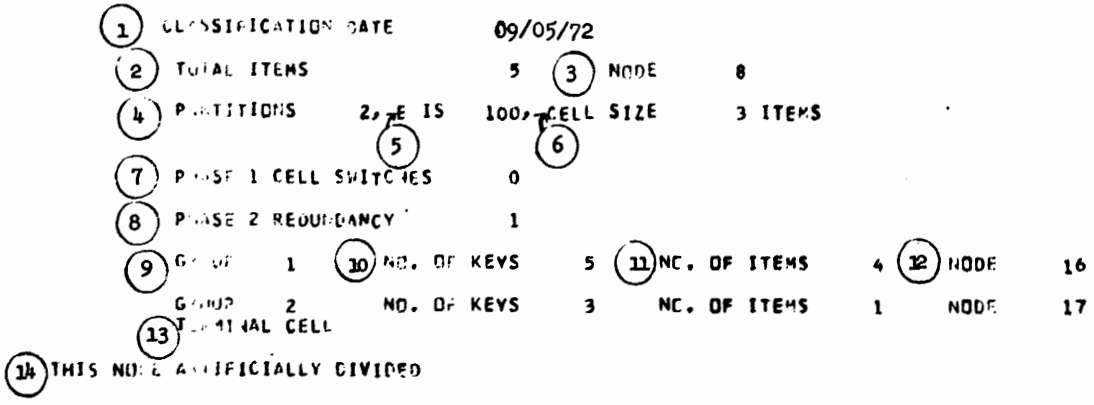

1 CLASSIFICATION DATE 09/05/72
2 TOTAL ITEMS 5
3 NODE 8
4 PARTITIONS 2, E IS 100, CELL SIZE 3 ITEMS
5
6
7 PHASE 1 CELL SWITCHES 0
8 PHASE 2 REDUNDANCY 1
9 GROUP 1
10 NO. OF KEYS 5
11 NO. OF ITEMS 4
12 NODE 16
GROUP 2 NO. OF KEYS 3 NO. OF ITEMS 1 NODE 17
13 TERMINAL CELL
14 THIS NODE ARTIFICIALLY DIVIDED

Figure 3.8

(8) Phase 2 redundancy is a counter of the number of document records that are not assigned to a group in PASS 2, but are instead written on the temporary document file for input to PASS 3.

(9),(10),(11) and (12) are summary indicators for the new groups formed in the current partition.

(9) identifies the group in the particular partition

(10) count of the no. of keys in the group

(11) count of no. of document records in the group

(12) Node number assignment for this particular group

(13) If the group formed is a terminal cell, then this factor is indicated underneath the group number.

(14) If the node named in item 3 has been artificially partitioned as explained in Section 3.2.1.1, this is indicated after all the summary information

3.2.7 Input Parameters

Table 3.7 indicates the fields on the input parameter card, that are necessary for running the CLASFY program.

3.2.7.1 Stratification Number

Stratification number, as was already indicated is the number of branches leading out of any one node in a tree. It is an input parameter because it is possible that every collection of documents might have a different optimum stratification number. It is up to the user to choose the optimum number for his collection.

There are 3 main points to consider when choosing a stratification number. First, what is the optimum number that will create a classification tree that is most suitable for browsing through the document

Field No.	Card Columns	Program Variable	Format	Description
1	1-2	NSTRAT	Integer	Stratification Number
2	4-7	IE	Integer	Sensitivity Factor
3	9-13	ICELSZ	Integer	Maximum Cell Size
4	15-19	NKPTKR	Integer	No. of keys per temporary key record
5	21-25	NKPPKR	Integer	No. of keys per final key record
6	73-80	IDATE	Integer	Current Date (MM/DD/YY)

TABLE 3.7

Input Parameters to CLASFY

collection. According to experiments by Thompson [10] the optimum stratification number for browsing always lies in the range of 3-5.

Second, one must consider the optimum number N, that will minimize the number of keywords per terminal cell, given a set number of terminal cells. That is, given a fixed number of terminal cells, fewer keywords per cell would be obviously more advantageous in as much as this would be an indication of less overlap of documents between cell groups. According to experiments by Litofsky [3] on a small document collection the optimum stratification number of 3 resulted in fewest keywords per cell.

The third point to consider is the size and type of the collection which will be classified. Though no experiments have been done, it might be an intuitive feeling on the part of the user, that his particular large document collection might be better classified with a larger stratification number. It should be remembered, however, that experiments have shown, that this larger N should most likely not lie above 5.

3.2.7.2 Sensitivity Factor

The sensitivity factor designated as E, is an important factor in the classification. As explained in section 3.2.1, E is the controlling factor which prevents the number of keywords in any one group from differing from any other group by more than the number E.

To consider the extremes, a very small E value will make the keyword groups even in size. According to Litofsky [3] the number of documents per group is about proportional to the number of keys per group for small document collections. Since a good classification

would have cells of approximately the same size, it would seem best to have a small E for small document collections. This ratio of documents per group to keys per group, is not however near one for large document collections. [3] Here it is necessary to consider the other extreme; a very large E value. A large E emphasizes keyword co-occurrence among the document descriptions. To explain this further, a very large E, would not allow the size of a group to be a factor in determining to which group the keywords of the document should be added. In other words, a group that was already large would not be prevented from acquiring the keywords of a document that did in fact have the largest co-occurrence of keywords. This emphasis on keyword co-occurence would indicate that for large document collections, a large E value should be used since the object of a good classification is to group like documents together.

However, there are two factors to consider when increasing E to some large value. First, the improvement in a classification by increasing E, decreases as E gets larger. [3] Second, if E is set too high, the size of the keyword groups could be greatly unbalanced and in the extreme case, an E that is too large could cause all keywords in the document collection to be placed in one group, at which time the node would have to be artificially partitioned as described in Section 3.2.1.1. With these facts in mind, E should be set to some value which would take advantage of the keyword co-occurrence factor while not excessively unbalancing the size of the groups.

3.2.7.3 Cell Size

This field is the criterion which determines a terminal cell. No

terminal cell can be larger than this input parameter. There are a number of factors to consider in the decision of size of this parameter. First, the cell size should be some multiple of the storage unit that will be used to store the cells containing the documents. Secondly, the following trade-off should be considered

1) a large cell size, will create fewer cells and the directory that will ultimately be created to access the cells will be smaller.

2) a small cell size will create more cells, with fewer documents per cell, which means less time to search through a cell.

Third, a small cell size, means fewer keys per cell resulting in a smaller selection with each cell access.

3.2.7.4 Number of Keys per Intermediate Key Record

The purpose of this input parameter (field 4 in table 3.7) is to take advantage [in terms of Input-Output time] of the computer system on which the classification is being run. The CLASFY program spends much time in Input/Output processing in that for each document in PASS1 and PASS2, each keyword file must be read in order to process the document. Since the keyword files can be rather large, and are accessed often, the most efficient method should be used to read and write these files. More efficiency is obtained by reading and writing large numbers of keywords at one time, rather than one keyword or a small number of keywords at one time. Each group of keywords is considered to be one record, and the number of keys in this record is determined by this input parameter. This parameter can be as large as the computer system will allow. That is, field 4 is bounded only by the maximum physical record size that the system (on which CLASFY is

being run), will allow. One must also take into consideration the storage device used, because different storage devices allow different maximum physical record sizes.

3.2.7.5 Number of Keys per Final Key Record

The purpose of this field (field 5 in table 3.7) is the same as field 4. However the number in this field should always be 5 keys smaller than the number of keys in field 4, if field 4 had been set to the maximum for the system. The reason why this field must be 5 keys smaller when field 4 is at the system maximum is to allow for the following:

1) Each key record on the Final Keyword File contains not only keys but also a node number (2 bytes), and a sequence number (2 bytes).

2) Each key group in the file has a header record associated with it which is 6 bytes in length.

The total of (1) and (2) above is 10 bytes of 5 keys (since each key is 2 bytes).

Since it is advisable to be able to fit the header record and the key record into one physical block on the file in order to attain the most efficient blocking, if field 4 is already set to the maximum block size then field 5 must by 5 keys or 10 bytes less. This method of packing many keys into one record is used in other programs in the system. It is used in Step 3, (TREE Program) and Step 5(NDTOKY Program). It will hereafter be referred to as the key blocking factor.

3.2.8 Restrictions on the CLASFY Program

There are two restrictions when running the CLASFY program.

1) The maximum stratification number is 9. This should not create

any burden on the user since as stated in Section 3.2.7.1, the optimum is between 3 and 5.

2) The maximum number of unique keywords allowed in the system after indexing is 32,767. This also should not be any problem to the user since it would be expected that any large file that was to be classified, would be homogeneous enough in nature to have the number of unique keywords less than 32,767.

If the user wanted to bypass either of these restrictions, small programming changes would be necessary. For 1) and 2), array sizes in the program would have to be changed. For 2), 4 bytes would have to be set aside to store keys on the input file, whereas only 2 bytes are used now.

3.2.9 Control Totals

An example of the control totals that are printed when CLASFY is complete is shown in Table 3.8. The 'number of records on the key file'(1) is the number of records actually written on the Final Keyword file. The 'number of levels in the tree'(2) is just the number of levels from top to bottom in the tree. Both of these totals are used as input in Step 3 to run the TREE program. The 'number of records on the document node file'(3) is self explanatory and should always be equal to the number of records that are being classified, because one record is written on this file for each document being classified. The 'number of nodes in the tree'(4) is just a control total to indicate the actual number of nodes in the classification tree. This figure should be compared to the control total in Step 3 which also gives number of nodes in the tree. These two totals should always be equal.

(1) No. of Records on Key File

(2) No. of Levels in the Tree

(3) No. of Records on Document-Node File

(4) No. of Nodes in the Tree

TABLE 3.8

CLASFY CONTROL TOTALS

3.2.10 Restart Procedures

Since the classification program will often be quite lengthy, it is suggested that the program be run in segments when and if the length of time needed to classify the complete file is too long to be done at once. (The length of time to run CLASFY, is dependent on the size of the file, the number of keywords for each record in the file, and the speed of the machine which is performing the classification). A logical time to stop execution of CLASFY is after a level in the tree has been completely partitioned or after completion of any partition. Since each partition is independent of the other it would then only be necessary to continue partitioning on the next node when CLASFY was restarted.

There is no restart facility presently included in the Classification System. However, it is suggested that the user investigate the system restart capabilities on the computer being used to carry out the classification.

The 'checkpoint and Restart' system package on the RCA Spectra 70/46 was used to classify the experimental data base described in Section 3.11 of this chapter. One complete level of the tree was completed on the experimental data base, a 'checkpoint' was taken, and the CLASFY program stopped. A checkpoint is a procedure used by the system which stores all necessary data and files that would be needed to 'restart' the program from where it left off. The 'restart' routines of the Spectra were then used to continue partitioning where it had stopped at the time of checkpoint.

3.3 Sort of Final Keyword File

The final keyword file when it comes out of CLASFY is in ascending node number sequence. Since the TREE Program (STEP 3 and described in the next Section) needs this file in descending node number sequence in order to create the hierarchical tree, this step in the classification process if performed. The final keyword file is sorted on fields one and two shown in Table 3.5. Field one is the terminal node number which is sorted in descending order. Each node could possibly contain two or more records. For example it must contain a header record, and it also will contain as many records as needed to hold all the keys associated with the node. The sequence number is therefore used in the sort to keep all the records for one node in the correct order. The sequence number on the header record would always be equal to one and would therefore always be the first record encountered in the node. All records in the node would be in ascending sequence number order.

3.4 Creation of a Hierarchical Tree

Each terminal node or cell in the classified file has associated with it a set of keywords which consists of the union of all keys in all documents in the cell. These terminal cells and keys are contained in the final keyword file which is output from the classification, as was already indicated. The classification which produced these terminal cells and associated keys was produced by starting at the top node of the tree with the union of all keys in all documents. Each node in the intermediate classification tree would contain all keys in any node which was descended from it in the tree. An example of such a tree is shown in figure 3.4.

One of the advantages of the classification of a data base is that a hierarchy structure is created in which sets of keywords that appear at each node of the tree are more specific than any set of keywords that is associated with any of its parent nodes in the tree. That means that broad categories are implied by the set of keywords at each node on the top of the tree while more specific categories are found further down the tree. This is a hierarchical tree that is suitable to browsing before retrievable from the data base.

The intermediate classification tree with its terminal nodes and associated keys created in CLASFY is not in this form. The tree program creates this hierarchical tree, by starting from the bottom of the classification tree with the last terminal node and intersecting all N sibling nodes of one parent. The set of keywords in the intersection then becomes the set of keywords for the parent node, and all common keywords are then deleted from the sibling nodes. This

intersection continues until the top of the tree (node 1) is reached. A tree is produced like the one in figure 3.9. Figure 3.9 contains the tree created as a result of running CLASFY for a set of documents. A comparison of figure 3.4 and figure 3.9 will help to explain the procedure that would be used to create the (hierarchical) tree, for the same set of documents. It is this tree that is now used for retrieval. In the sections to follow, this tree will now be referred to as the classification tree, and the tree structure which is first constructed in CLASFY will be referred to as before as the intermediate classification tree.

From this generation of the tree it seems obvious that the more frequently a term is used, the higher it would appear in the hierarchy. For example, a word that appeared in all documents would rise to the top node. In considering how far up the tree one might expect to find a keyword, one must be careful to consider not only the frequency of occurrence of the key term, but also the number of documents it appeared in. A keyword that occurs frequently in a specialized group of documents would not be found high in the hierarchy.

An important factor to note in considering the classification tree in retrievals, is that the keyword hierarchy comes about a posteriori. That is, the classification tree is built as a result of the document descriptors. It is not the case, that the hierarchy is used in order to obtain descriptors for documents.

A few points about this a posteriori hierarchical tree are important.

First, it is a property of the tree that every document contained in the file for which this tree is used is described by a set of

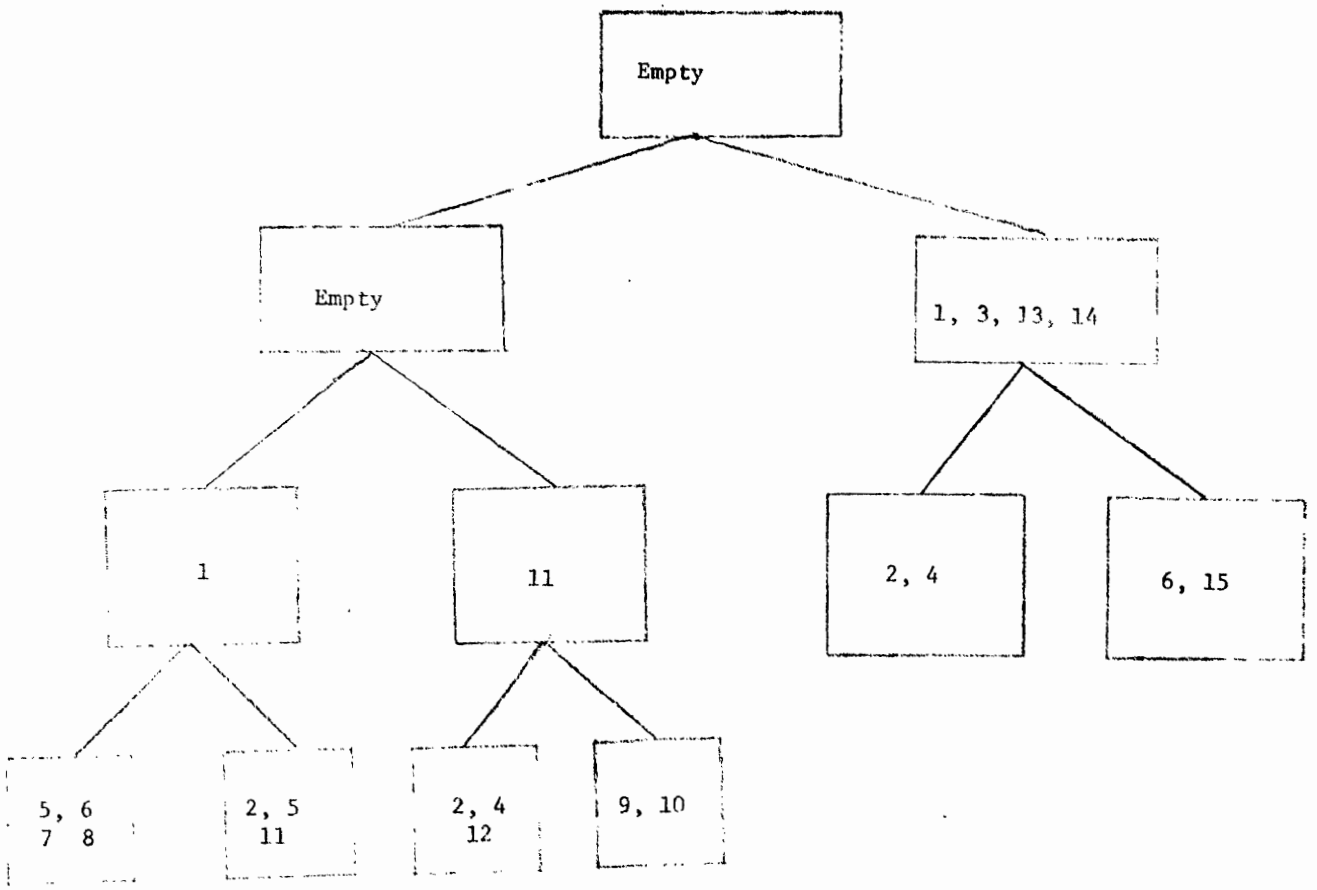


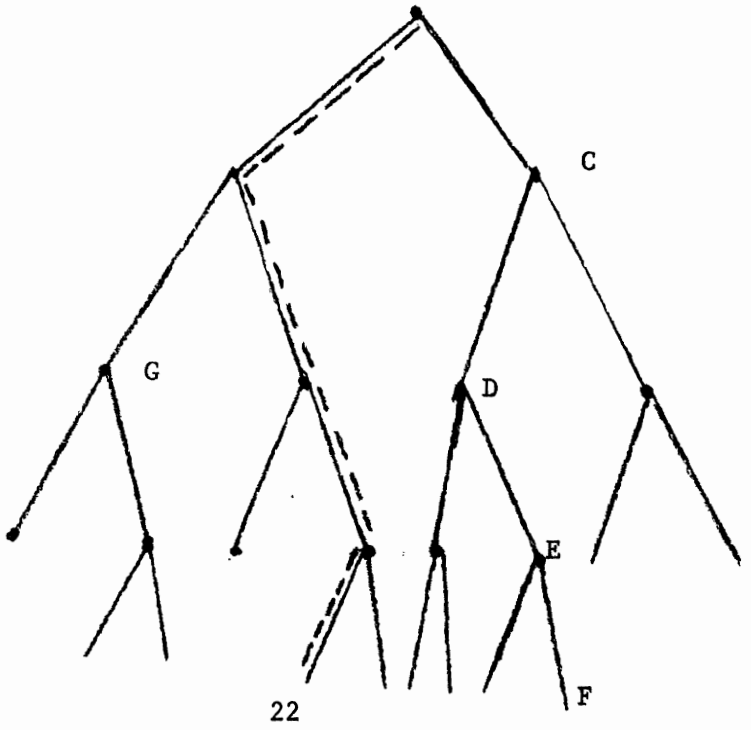
Figure 3.9

Hierarchical Classification Tree
For Example in Sec. 3.2.2

surrogates which is completely contained within a set of nodes that form a path in the tree from the top to the terminal node which contains the document. It is certain that one can find all the keywords of a document in the path A in Fig. 3.10 if the document being considered is in node 22 in the figure.

The second property is that each keyword will appear only once within the set of nodes in the path. (This is not to be confused with finding a keyword at more than one node. This can and does happen often). In term of fig. 3.10, if a keyword appeared at point C, it would not appear at nodes D, E, or F, but it could occur at point G.

The true hierarchy can be examined to judge somewhat the goodness of a classification. It appears that if many keywords appear above the bottom level of the tree, the algorithm did not do a very good job. This is because if all "like" documents were grouped together in one cell according to their keyword descriptors, we would not expect to find documents in other cells with the same descriptors. That is, in the ideal situation all documents with the descriptor 'X' would fall in one cell. Then in creating the tree, when intersecting the N sibling nodes, only one of the N nodes would contain 'X' and therefore 'X' would not rise in the hierarchy. In this ideal case all keywords in the tree would be at the bottom or cell level. But in any real situation, this would not happen. There will always be in a collection of documents, keywords that are common to enough segments of the collection to form a true hierarchy. However, the fewer the keywords above the bottom level, the better the



PATH A -------

3.10

Path in Classification Tree

classification.

3.4.1 TREE Description

As was noted in the last section, the TREE program must intersect all the nodes created in CLASFY, to create the hierarchical tree. To do this, the TREE program utilizes the fact that the intermediate tree created in CLASFY is a simulated balanced tree. The hierarchical tree is created in the following way:

The TREE program starts with the last N terminal nodes created. These N nodes are intersected. The keys in the intersection are deleted from the N nodes, and the nodes are written to the classification tree file. However, the keys in the intersection that form the parent node must be saved because as the intersecting progresses up the tree, this parent node will be intersected with other N-1 nodes to again get another parent node. In terms of fig. 3.11 nodes 35, 36, and 37 are intersected to form node 12. The keys of node 12 must be saved so that they can be intersected with the keys of nodes 11 and 13 to form the parent node number 4. The balanced tree numbering comes into effect here. Because the tree numbering is balanced, the following formula can be used to calculate the parent node of any given node number:

$$P = \mathrm{INT}\left|\frac{S-2+N}{N}\right|$$

where P = the parent node

S = the node number whose parent we are seeking

N = the stratification number of the tree being intersected

INT = Integer

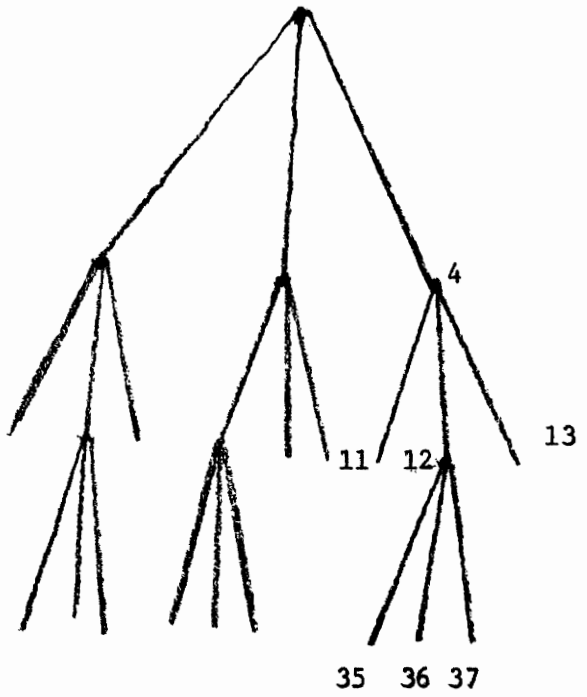


Fig. 3.11

Intersection to Form Classification Tree

For node 36 for example,

$$P = \frac{36-2+3}{3} \quad \text{INT} \quad \frac{37}{3} = 12$$

When nodes 35, 36, and 37 are intersected, these nodes (with common keys deleted) are written to the classification tree file and the keys in the intersection are saved as node number 12 on a temporary file for future intersection with nodes 11 and 13. All other nodes on the bottom level must be intersected before nodes 11, 12, and 13 will be accessed and intersected to create node 4. This routine continues until node 1 is finally reached.

3.4.2 Input Files

The final keyword file produced in CLASFY and then sorted in descending node number order is input to the tree program. It contains the same fields as described in Section 3.2.4 and shown in table 3.5.

3.4.3 Intermediate Files

There are 2 types of intermediate files used in the TREE program.

3.4.3.1 Intermediate Intersection Files

The number of these intermediate files is equal to N, the stratification number for the classification. These files are used during the intersection of the N sibling nodes to create their parent. During the intersecting process (since all N nodes are not intersected at one time), the nodes that have been processed are saved on these intermediate files so that when the intersection of the

N nodes is complete, the common keys can be deleted from each of the N nodes involved in the intersection.

3.4.3.2 Intermediate File For Future Intersection

As we explained in Section 3.4.1, when the N sibling nodes are intersected to form a parent node which contains the common keys of the N nodes, the parent node must be saved for future intersection as the processing progresses up the tree. These intermediate files (there are two) contains all the parent nodes which must be saved for future intersections.

3.4.4 Output Files

The output of the TREE program contains the classification tree that is described in Section 3.4. That is, it contains the nodes of the hierarchical tree followed by the keys associated with the node. The fields of this file are the same as those described in Section 3.2.4 and shown in table 3.9. The sequence of this file is descending node sequence.

3.4.5 Input Parameters

Table 3.10 indicates the fields in the parameter card that are necessary for running the TREE program.

1) Field 1 is the stratification number and is the same number as was used in CLASFY in step 1.

2) Field 2 is the number of levels in the tree that was created in CLASFY and is the number which is printed at the end of the CLASFY run as a control total (see Section 3.2.9)

Field No.	Card Columns	Program Variable	Format	Description
Header Record				
1	2	INODE1	Integer	Node Number in tree.
2	2	ISEQ	Integer	Sequence No.
3	2	INOKY1	Integer	No. of keys in the Node
Keyword Record				
1	2	INODE1	Integer	Node No. in tree
2	2	ISEQ	Integer	Sequence No.
3	2	IARRY1	Integer	Integer Key
. . . Maximum allowed				

TABLE 3-9

Classification Tree

Field No.	Card Columns	Program Variable	Format	Description
1	1-3	NSTRAT	Integer	Stratification Number
2	5-7	ILEVEL	Integer	No. of levels in the intermediate classification tree
3	9-14	INOREC	Integer	No. of records on the Final Keyword File
4	16-20	NKPREC	Integer	No. of keys per record on the Classification Tree File
5	22-26	NKPSCR	Integer	No. of keys per record on the temporary Key Files

TABLE 3.10

Input Parameters to the Tree Program

3) Field 3 is the number of records on the Final Keyword file which in sorted form is input to the TREE program. This number is also found at the end of the CLASTY run as a control total (see Section 3.2.9).

4) Field 4 is the number of keys per record on the Classification Tree file which is the output for the TREE program and also the number of keys per record on the Final Keyword file which is input to the TREE program. The purpose of this field as described in Section 3.2.7.5 is to have many keys in each record on the file instead of one key per record. It is the key blocking factor that was referred to in Section 3.2.7.5. This parameter must be equal to field 5 in table 3.7. That is, the final keyword file must be read in the same way that it was written. Since field 5 in Table 3.7 was used to control the number of keys per record when writing the file, this field which is used to read the file should be the same. And since the classification tree file is in the same format as the final keyword file, this field can also be used to control the number of keys in each record of the classification tree file.

5) Field 5 is the number of keys per record on the Intermediate Files used for Intersection. Again this field is used for packing many keys in to one record and its description is analogous to the description given in Section 3.2.7.4, and should be the same value.

3.4.6 Control Totals

An example of the control totals that are produced when the TREE program is complete is shown below. The 'number of records on the tree file' (1) is a count of the number of records on the output

classification tree file. The 'number of keys in the trees counting duplicated', (2) is the sum of all the keys at each node in the tree. Both of these control totals are provided for the user for general information purposes, and may be disregarded if the user finds them of no interest. 'The number of nodes on the tree file' (3) is precisely the number of nodes in the classification tree that was created by the TREE program. It must be equal to the control total (4) in Table 3.8 which is output from CLASFY. These two totals should be compared before the next step is executed.

(1) Number of records on Tree File

(2) Number of keys in Tree counting duplicates

(3) Number of nodes on the Tree File

3.5 Sort of the Classification TREE

The classification tree when it is produced from the TREE program is in descending node sequence. Since the input to create the Node-to-key dictionary in the NDTOKY program (Step 5 in the classification system and described in the next section) requires the file in ascending node number sequence, this step in the classification process is performed. The classification tree is sorted on fields one and two shown in Table 3.10. Field one is the node number of the tree which is sorted in ascending order. Again as in the sort of the final keyword file the sequence number in field 2 is used to keep all records for one node in the correct order.

3.6 NODE-TO-KEY TABLE

The node-to-key table (which is merely a list of nodes and keys belonging to the nodes) that will be discussed in this section, is an

aid to the user in browsing through his document collection. It is this file, and the file generated in Step 7 which will be referred to as the dictionarys of the classification. The classification tree created in Step 3 is the tool used to accomplish this. The classification tree shown in Fig. 3.9 can be thought of as a set of nodes, each node consisting of a set of keywords with more generic node descriptions on the top of the tree, and more specific node descriptions toward the bottom. All the keywords in a path of a tree can be thought of as an abstract of the knowledge contained beneath the top node in the path [6].

How might a user browse through his file? First, all browsing requires a hierarchical classification so that one can start with a broad term and if desired, go on to more specific categories. This is possible with the hierarchical classification tree. The user might have a conjunction of keywords and enter some node in the tree. The response he would require from the system, could be the display of the nodes beneath the original one along with statistics such as how many documents in these nodes contain the keywords he is interested in, as well as which nodes beneath the original contain the keywords he wants. The user would then decide to which node he would like to proceed to next after looking at these statistics.

For example, one could enter the system at node 1.1 in Fig. 3.12 and continue to node 1.1.1 or 1.1.2 after analyzing the statistics. [In this Figure the canonical method of node numbering is used as described in Section 3.2].

To achieve this browsing capability, given any node, the keywords

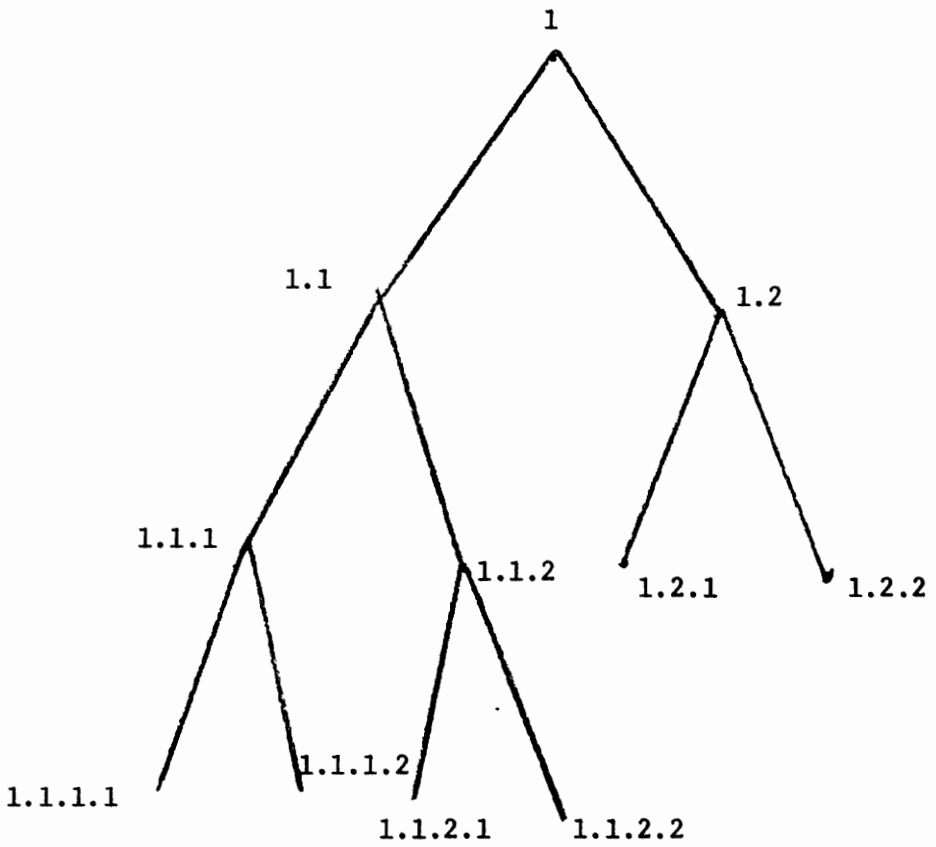


Fig. 3.12

Canonical Method of Node Numbering

associated with it must be easily found. This is done by using the Node-to-key Table described in the following sections.

3.6.1 NDTOKY (Node-to-Key) Description

The 'NDTOKY' program builds a table which consists of all nodes in the classification tree along with its associated alphabetic keys.

The Node-to-key table is really a representation of the tree hierarchy created in Step 3. The only difference is the substitution of the alphabetic keys for the integer keys.

The program reads each node record on the sorted classification tree file and looks up each integer keyword to obtain the alphabetic representation. The lookup is done on the file of unique keywords which was described in Section 2.82 of the indexing.

An actual printed Node-to-key table is produced as well as a physical file. It is this file that is used in browsing. With this file, given any node, its alphabetic keywords can quickly be displayed.

In processing each node in the tree, another file is created which contains a record for each key in the tree along with its associated node number. The key is alphabetic. This file produced will be used in step 7 and will be explained in Section 3.6.3.

3.6.2 Conversion to Canonical Node Number and Back Again

Input to the NDTOKY program contains integer node number while the printed report contains canonical node number. The output file contains integer number and in browsing and retrieval canonical number will be needed at times. There will also be cases when canonical node number will be given and integer node number will be needed for lookup in the two dictionary files of this system. Both dictionary files store the

node numbers as integers and not in their canonical form in order to save space.

Therefore it is necessary to be able to convert from canonical node to integer node and vice versa. Here again the simulated balanced tree numbering makes the following conversions possible.

1) Conversion from canonical node number to integer node number is done as follows:

$$I = aN^0 + bN^1 + cN^2 + dN^3 +,..ZN^{(y-2)} + 1$$

where

I = integer node

N = stratification number

a, b, c,...Z = the digits in the canonical node number from right to left excluding 1^{st} digit on the left.

Y = the level at which the node is found

For example, where N=3

$$1.1.3.3 = aN^0 + bN^1 + cN^2 + 1$$
$$= 3(3^0) + 3(3^1) + 1(3^2) + 1$$
$$= 3 + 9 + 9 + 1$$

1.1.3.1 = 22

2) Converting integer node number to canonical node number is done in the following way.

a) Each character in the canonical node number from right to left is calculated separately with the exception of the 1^{st} digit on the left which is always known to be one.

The calculation is as follows:

$$X_i = \text{Mod}([N_j - C_j], \text{NSTRAT}) + 1$$

where X_i is the digit in the canonical node

NSTRAT - is the stratification number of the tree

N_j - for the 1^{st} X_i, it is the node being converted; for each X_i to the left, N_j is the parent node calculated from the current N_j

C_j - the value of the left most node of the tree at the level at which N_j is found

3.6.3 Input to NODE-TO-KEY Program

There are two files used as input to the 'NDTOKY' program. Both of them have already been previously described. The first is the UNIQUE-WORD/CODE file which is described in Section 2.82 of the indexing and has the fields as illustrated in Table 2.14. This is the file that is used to lookup the alphabetic keywords given an integer keyword. The other file is the sorted classification tree which is illustrated in Table 3.9. This is the file that is used to actually produce the node-to-key dictionary.

3.6.4 Output Files

There are two output files in the NDTOKY program

1) The first is the alphabetic key and node file which contains a record for each key contained in the classification tree. The fields in the records in this file are illustrated in Table 3.11. Field one is the integer node number at which the key was found, and field 2 is

Field No.	Length in Bytes	Program Variable	Format	Description
1	2	INODE	Integer	Node Number
2	40	IWD	Alphabetic	Keyword

TABLE 3.11

Alphabetic Key-Node File

the alphabetic key itself. This file is used in Step 7 to produce the KEY-TO-NODE Dictionary.

2) The second file produced in this program is the Node-to-Key dictionary. It has the fields as described in Table 3.12. Each node in the classification tree has associated with it on this file a header record indicating the integer node number, and the number of keys found in this particular node, and as many keyword records as it takes to hold all the keys in this node. It may be pointed out that the keywords are alphabetic. The maximum number of keywords contained in each record, or the keyword blocking factor is an input parameter and is described in Section 3.6.5.

This file is the Node-to-key dictionary that is used in browsing and retrieval.

3.6.5 Input Parameters

Table 3.13 indicates the fields in the parameter card that are necessary for running the NDTOKY program.

1) Field 1 is the number of keys per record or the key blocking factor for the sorted classification tree file (input to NDTOKY). The purpose of this field again as described in Section 3.2.7.5 is to pack as many keys as possible into one record. This must be equal to field 5 in Table 3.7 and field 4 in Table 3.10. It is again reiterated, that these parameters must be constant in the three programs, CLASFY, NDTOKY, and TREE.

2) Field 2 is the number of levels in the classification tree. It is the number which was printed at the end of the CLASFY run in the control totals, and also used as input to the TREE program.

Field No.	Length in Bytes	Program Variable	Format	Description
Header Record				
1	2	INODE	Integer	Node Number
2	2	INOKY	Integer	No. of keys in the node
Key Record				
1	2	INODE	Integer	Node Number
2	40	INDARY	Alphabetic	Alphabetic Key
.				.
.				.
.				.
maximum allowed				maximum allowed
		.		
		.		

TABLE 3.12

Node-To-Key Dictionary

Field No.	Card Columns	Program Variable	Format	Description
1	1-5	IBLF	Integer	Key Blocking Factor for Classification Tree File
2	6-9	ILEVEL	Integer	No. of levels in the Classification Tree
3	10-12	NSTRAT	Integer	Stratification No.
4	13-15	MAXWL	Integer	Maximum No. of Characters in any key term
5	16-18	NKPREC	Integer	Key Blocking Factor for Node-to-Key Dictionary

TABLE 3.13

Input Parameters to NDTOKY

3) Field 3 is the stratification number and is the same as was used in CLASFY and TREE programs.

4) Field 4 is the maximum word length for any index term. As was already described in the indexing chapter, index terms must have some maximum length. This value in the NDTOKY program should be the same value as that used in the indexing routines. This parameter is needed in this program because the NDTOKY program works with the alphabetic keywords.

5) Field 5 is the number of keys per record in the NODE-TO-KEY dictionary file. Again this parameter is a key blocking factor used for the same purpose as previously described in Section 3.2.7.5, but it must be of a different size than the one used in field 1. Field one is used to pack integer keys into a record whereas this field packs alphabetic keys into a record. One must consider the length of the keys in order to calculate this parameter. If a keyword has a maximum length of 20, and each character in the key is 2 bytes, that means that one alphabetic keyword takes up 40 bytes. With this figure known, the number of keys that can fit into the largest allowable record can be calculated for the computer system being used.

3.6.6 Node-To-Key-Table Output Report

A sample of the NODE-TO-KEY Table output report is shown in figure 3.13. This report contains exactly what is on the Node-to-Key dictionary file except that the keywords on the report are printed in canonical form. It is the dictionary file that is most important, and used automatically in the retrieval system, but this report can be

Node	Key
1	Empty
1.1	Empty
1.2	1, 3, 13, 14
1.1.1	1
1.1.2	11
1.2.1	2, 4
1.2.2	6, 15
1.1.1.1	5, 6, 7, 8
1.1.1.2	2, 5, 11
1.1.2.1	2, 4, 12
1.1.2.2	9, 10

Fig. 3.13

Node-To-Key Table for Example

used for quick reference. It contains each node in the classification tree in canonical form, and all keywords found in the node.

3.6.7 Control Totals

The totals given at the end of the NDTOKY run and shown below are strictly for the user's information and serve no definite purpose. The 'number of nodes printed'(1) is the number of nodes listed in the NODE-TO-KEY Table. The 'number of key records written'(2) is the number of records that were written on the alphabetic key node file.

(1) Number of nodes printed

(2) Number of key records written

3.7 SORT OF ALPHABETIC KEY AND NODE FILE

The alphabetic key-node file (Table 3.11) is created in the NDTOKY program. The records, (upon output from NDTOKY) are in no special order. Also there are duplicate keys contained on this file since if a key is found at nodes 22 and 25, there would be two records for that key not necessarily together on the file. This file is used as input to the KYTOND program which builds the KEY-TO-NODE dictionary. This can only be done if this file is sorted by alphabetic keyword as a major key, and node number as a minor key. This will group together all records for one alphabetic key, and the nodes in the records will be in ascending order. This will make it possible for the KYTOND program to easily obtain all nodes at which a key is found.

3.8 KEY-TO-NODE DICTIONARY

The key-to-node dictionary is a file of all the unique alphabetic

keywords in the document file, with each keyword followed by the nodes it can be found at. This dictionary is an aid in retrieval. To illustrate its use, suppose that a user is interested in all the documents that contain the keywords cat, dog and pet. The user presents these keywords to the retrieval system as a conjunction of keywords:

cat and dog and pet

These keywords would first be converted to integer keywords. For the purpose of this example suppose the above request converted to:

5 & 2 & 7

Then using the key-to-node dictionary all nodes containing these keys could be obtained. The nodes must be translated to canonical numbering because the canonical form is used in the next step of this example.

After the nodes are obtained from the dictionary, the query could look as follows:

1.2 & 1.1.1 & 1.2.1

These canonical nodes are then used in determining valid search paths [2]. By valid search paths, is meant a path in the tree which would lead to a cell containing documents that have all the keywords asked for in the request. It is possible that no such path exists. That means that no document in the file contains all the keywords in the query, and the query would have to be modified.

Once valid search paths are found the documents themselves could actually be retrieved by searching the terminal cells at the end of the search paths [2]. Or perhaps the user might be interested in seeing the other keys at the nodes which are on the top of the search

path. In this case the node-to-key dictionary can be utilized to give the display of the keys at these nodes.

Therefore, to aid in this function of retrieval by conjunction or disjunction of keywords, as in the example above, the key-to-node dictionary is supplied in this system.

3.8.1 KYTOND (KEY-TO-NODE) DESCRIPTION

The KYTOND program builds a dictionary that consists of all unique alphabetic keywords in the data base.

In step 5 in the NDTOKY program a record was created for each key containing the key and its node. This file is then sorted in Step 6 by alphabetic key. Then the KYTOND program reads each key and creates one record for each unique key followed by all its nodes. The nodes are stored as integer node numbers and are converted for use in retrieval. A printed KYTOND table is produced as well as a physical file called the dictionary. It is this file that is used for retrievals. With this file, given any key, all nodes in the tree that contain the key can quickly be found.

3.8.2 Input Files

The sorted alphabetic key-node file is the input to the KYTOND program. It has the fields shown in Table 3.11, and already described. This file produces the KEY-TO-NODE dictionary.

3.8.3. Output Files

The output file produced in this program is the KEY-TO-NODE dictionary. It has the fields as described in Table 3.14. Each unique key in the classification tree has associated with it on this file, a header record, indicating the alphabetic key and the

Field No.	Length in Bytes	Program Variable	Format	Description
Header Record				
1	40	IWD	Alphabetic	Alphabetic Keyword
2	2	INDE	Integer	No. of Nodes at which keyword is found
Node Record				
1	40	IWD	Alphabetic	Alphabetic Keyword
2	2	NDARY	Integer	Integer Node No.
.				.
.				.
.				.
maximum allowed				maximum allowed

TABLE 3.14

Key-To-Node Dictionary

number of nodes at which this particular key can be found, and as many node records as it takes to hold all the nodes associated with this keyword. The nodes are stored in integer form for the sake of space. These integer nodes are converted if necessary when using the key-to-node dictionary.

The maximum number of nodes contained in each record after the header is the node blocking factor, and is an input parameter described in Section 3.8.4. Each node record contains the key itself plus some number of integer nodes.

This file is the KEY-TO-NODE dictionary that is used in retrieval.

3.8.4 Input Parameters

Table 3.15 indicates the fields in the parameter card that are necessary to run the KYTOND program.

1) Field 1 is the maximum number of nodes per record in the KEY-TO-NODE dictionary file. It is a blocking factor similar to the one described in Section 3.2.7.5. In that case keys were being packed into a record; in this case nodes are being packed into a record. This field must be calculated for the system on which the KYTOND program is run, keeping in mind the size of the node (2 bytes) and the maximum allowable record size.

2) Field 2 is the number of levels in the classification tree. It is the number which was presented at the end of the CLASFY run in the control totals.

3) Field 3 is the stratification number used in CLASFY.

4) Field 4 is the maximum word length for any index term. It must be the same value as that given in the indexing routines and in

Field No.	Card Columns	Program Variable	Format	Description
1	1-5	IBLF	Integer	Key Blocking Factor for Key-to-Node Dictionary
2	6-8	ILEVEL	Integer	No. of levels in Classification Tree
3	9-11	NSTRAT	Integer	Stratification No.
4	12-14	MAXWL	Integer	Maximum No. of characters in any keyword

TABLE 3.15

Input Parameters for Key-To-Node Program

the NDTOKY program.

3.8.5 KEY-TO-NODE Table Output Report

A sample of the KEY-TO-NODE Table output report is shown in Figure 3.14. This report contains exactly what is on the KEY-TO-NODE dictionary file. It is the dictionary file that is used automatically in a retrieval system, but this report can be used for quick reference. It contains each unique keyword in the classification tree followed by all nodes associated with the key. Although, the dictionary contains the nodes in integer form, these nodes are translated to canonical form before printing.

3.8.6 Control Totals

There is only one total given at the end of the KYTOND run and that is the number of unique keys in the tree. This total should match the number of unique keys calculated in the indexing routines.

3.9 Sort of the Document Node File

The document-node file as described in Section 3.2.4.2 with fields as displayed in Table 3.6 contains a document number along with its node assignment. When this file is created in CLASFY it is in node number order. However when it is used in the MRGCLY program (step 9 in the classification system and described in the next section) it is needed in document number order. So this file is sorted on field one, the document number and is then used as input in the next step.

3.10 Creation of a Final Classified File

When the classification is complete in step 1 of the classification system, all documents are assigned to a node. The document surrogate

Key	Node
1	1.2, 1.1.1
2	1.2.1, 1.1.1.2, 1.1.2.1
3	1.2
4	1.2.1, 1.1.2.1
5	1.1.1.1, 1.1.1.2
6	1.2.2, 1.1.1.1
7	1.1.1.1
8	1.1.1.1
9	1.1.2.2
10	1.1.2.2
11	1.1.2, 1.1.1.2
12	1.1.2.1
13	1.2
14	1.2
15	1.2.2

Fig. 3.14

Key-To-Node Table for Example

file as explained in Section 3.2.2 contains all documents with their cell assignments. However, since the document surrogate file consists of document number along with integer representations of the alphabetic keys assigned to the document, it is necessary to transform the classified document surrogate file into some useable form which could be considered to be the classified data base. The classified data base would obviously consist of more than just document numbers and keywords. The full text of the documents, the alphabetic keywords, and cell number assignment should all be included in order for it to be useful in a retrieval system. The program that is described in the next section carries out this task.

3.10.1 MRGCLY - (Merge of the Classified File) Description

The MRGCLY program creates a classified file with all necessary components; text, keywords and cell assignment. This is done by merging three different files (created in this system) into one classified file.

The first file used is the original document text file (standard input file) that was input to the first step of the indexing process. This file contains document number and text. The second file used is the Unique-word-within-document file created in Step 4 of the indexing. As was already mentioned in Section 2.6.2 this file contains the document number and all alphabetic keywords assigned to this document. The last file used is the document node file that was created as output of CLASFY and described in Section 3.2.4.2. It contains a document number and associated cell assignment. All three are in document number order upon input to this program.

The merge is accomplished by first reading the document text file, then reading the document keyword file to obtain all keywords for the document and finally reading the document node file and using it to look up the cell number assigned in CLASFY to this particular document. Then all the pertinent information is written to the Classified Text file. It is this file which will be used in the retrieval system for this data base.

3.10.2 Input Files

Three files, mentioned in the last section and already described, are used as input.

1) Document Text File (Standard Input File) - Fig. 2.0

2) Document Node File - Table 3.6

3) Unique-Word-Within Document - Fig. 2.10

3.10.3 Output Files

One file is produced as a result of running the MRGCLY program. This file is the final classified file and has the fields shown in Table 3.16. This file has two kinds of records. One, contains the document text. There are as many of these records as it takes to contain the complete text. The other kind contains the document keys. There are as many of these records as it takes to contain the keys for a document.

3.10.3.1 Text Records

1) Field 1 - Length of the text contained in the record

2) Field 2 - Document Number

3) Field 3 - Node (or cell) number to which this document is

Field No.	Length in Bytes	Program Variable	Format	Description
1	2	LEN	Integer	Length of text to follow
2	4	NKDN	Integer	Document Number
3	2	IMODE	Integer	Terminal Node No.
4	2	INEXT	Integer	Record Indicator 0 - more of same type of record to follow 1 - record of different type to follow
5	2	ISEQ	Integer	Sequence Number
6 • • • 2042	2	ITEX	Alpha-Numeric	Text of Document • • • Up to 2036 characters in a record

TABLE 3.16a

Final Classified File - Text Record

Field No.	Length in Bytes	Program Variable	Format	Description
1	2	NOKYPR	Integer	No. of keys in record
2	4	NKDN	Integer	Document Number
3	2	INODE	Integer	Terminal Node Number
4	2	INEXT	Integer	Record Indicator 0 - more of same type of record to follow 1 - record of different type to follow
5	2	ISEQ	Integer	Sequence Number
6 . . . maximum allowed	40	IWORD	Alphabetic	Alphabetic Key . . . up to maximum number allowed in a record

TABLE 3.16b

Final Classified File - Key Record

assigned

4) Field 4 - Indicator for signifying whether more of the same type of record will follow

0 - indicates more to follow

1 - indicates the last of this type

5) Field 5 - Sequence number used to keep all records associated with one document in order. Text records will always come before key records.

6) Field 6 - Document Text with the length determined by field 1

3.10.3.2 Key Records

1) Field 1 - Number of keys contained in this record

2), 3), 4) - same as for Text Records

5) Field 5 - Alphabetic keys with the number in the record determined by field 1.

3.10.4 Input Parameters

The input parameters necessary to run the MRGCLY program are illustrated in Table 3.17.

1) Field 1 - Maximum length of any key term. Its value is the same as what had been throughout this system.

2) Field 2 - Number of keywords per key record. This parameter is again a key blocking factor and must be calculated keeping in mind the size of the keyword and the maximum length of a record allowed on the computer system being used.

Field No.	Card Columns	Program Variable	Format	Description
1	1-3	MAXWL	Integer	Maximum number of Characters for any keyword
2	4 8	NOKYPR	Integer	Key blocking factor for final Classified File

TABLE 3.17

Input Parameters for Creation of Final Classified File

3.11 CLASSIFICATION SUMMARY

The following section will attempt to give the user a step by step summary of the procedure which will be used when classifying any given data base. This summary can be used as a quick reference when using the classification system. Also, in the following section will be examples of the results obtained in the classification of an experimental data base. To perform a complete classification on a file, 9 programs or steps must be executed.

Each section to follow will contain a macro flow chart for the step being discussed, a quick reference chart to aid the user in locating pertinent information in this manual, which will facilitate execution of the step, and the control language used to execute the described program on the experimental data base. This control language includes the actual input parameter card used (if any) for the data base. All classification programs were run on the RCA Spectra 70/46 and the control language shown in the following examples is the control language of that machine. Since the programs in the classification are machine independent, only the control cards would need change in order to use the system on another machine.

As has already been explained, in the summary section on indexing, an experimental file containing the text of 1669 radio broadcasts was indexed using the semi-automatic indexing method explained in this manual. When the last step in the indexing was complete, the file called the standard input file or document surrogate file was ready to be used as input to the first step in the classification. The file has on it a record for each of the 1669 documents with each record containing the

document number and its associated keywords. The keywords as explained in the indexing, are integers, with each integer representing an alphabetic keyword.

Notes on the Following Sections

(1) The macro flow charts for each step contain numbers in parenthesis next to each file. These numbers are the Fortran logical file numbers used in the actual programs and must be the numbers used on the job control language for the specified files. In the job control language for the RCA Spectra 70/46 these numbers can be found in the 'LINK=DSET' parameter of the file control cards as shown in the job control language examples.

(2) The job control shown in the examples, assume that the programs at each step have been compiled and are in some secondary storage area ready to be executed.

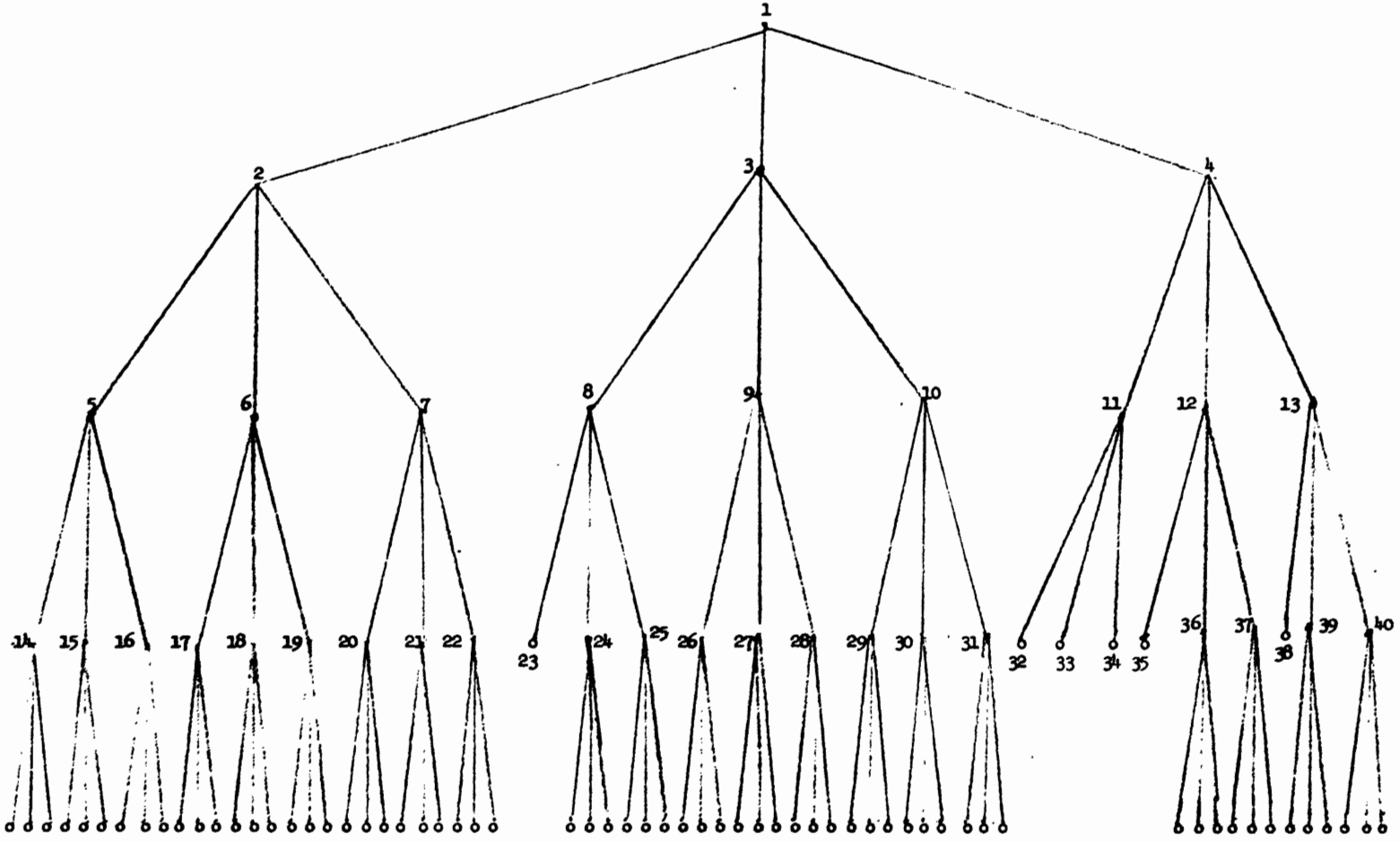
(3) Detail Flowcharts of the programs used in the classification are also included in the following sections.

3.11.1 STEP 1 CLASSIFICATION

PROGRAM NAME: CLASFY

OBJECTIVE: To automatically classify the document surrogate file

The 1669 documents were classified in this step. The intermediate classification tree created in this process is shown in Figure 3.15. The stratification number used was three, sensitivity factor was 75, and the last page of the classification report is illustrated in Figure 3.16. A larger sample can be found in the appendix. The total classification time using the RCA Spectra 70/46 was 32 hours, with the first partition on the 1669 documents taking 14 hours. Although the data base is not very large, the lengthy classification time is partially due to the large number of keywords associated with each document. The user is referred to Section 3.2.10 which discusses the execution of CLASFY in segments when run time is extremely lengthy. Each document could be characterized by as many as 250 keywords, but on the average, the number was 100 keywords per document. The final keyword file produced contained the keywords for each of the 69 terminal cells created in this classification. The key blocking factor used was 1000 resulting in 144 records on this file. The average number of keys in any terminal cell was 624.

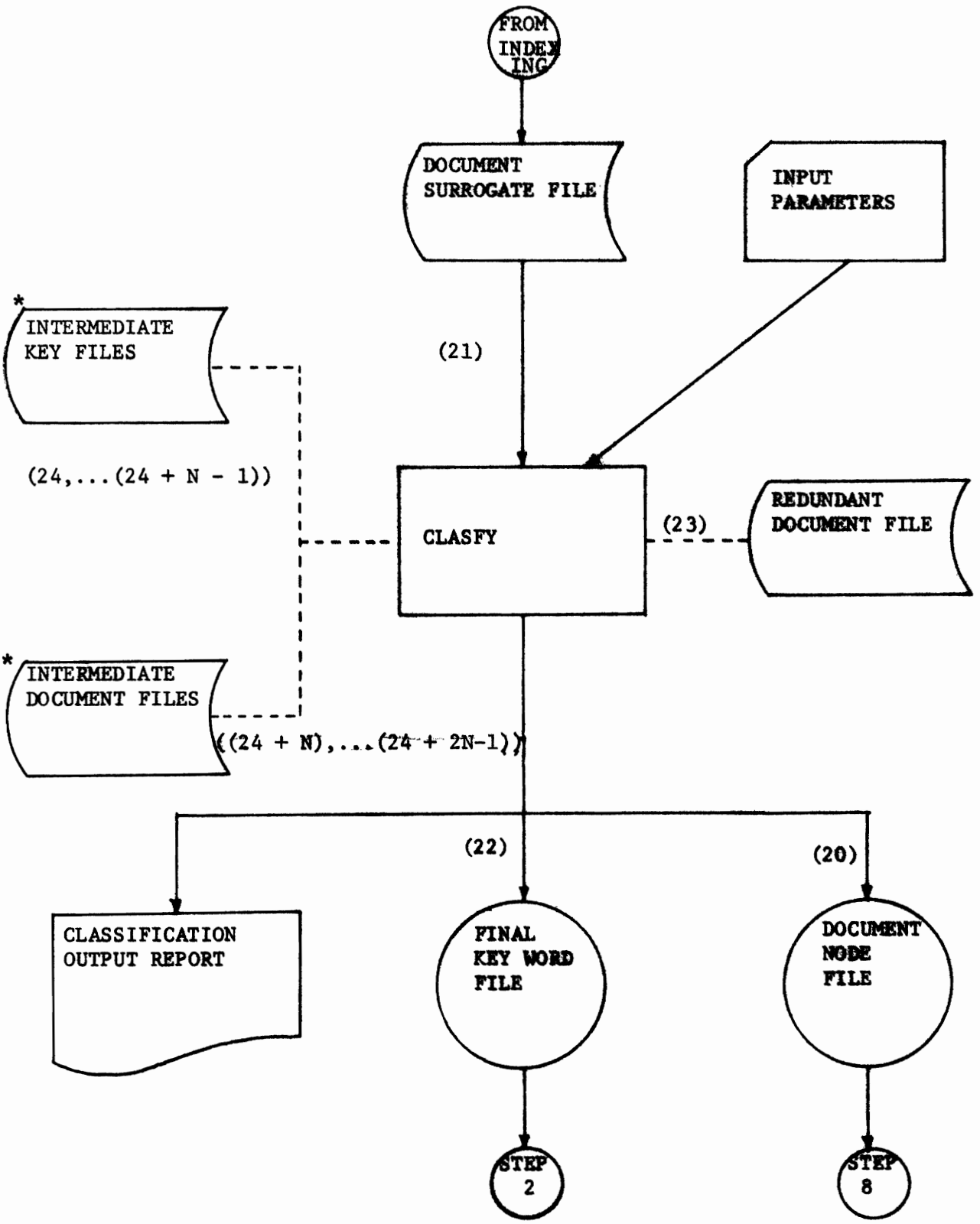


o indicates terminal nodes

Figure 3.15

```
CLASSIFICATION DATE          0
 TOTAL ITEMS                78        NODE      40
 PARTITIONS      3, E IS    75, CELL SIZE      50 ITEMS

 PHASE 1 CELL SWITCHES      19

 PHASE 2 REDUNDANCY          0

 GROUP    1      NO. OF KEYS   519     NO. OF ITEMS    23     NODE    119
 TERMINAL CELL

 GROUP    2      NO. OF KEYS   588     NO. OF ITEMS    20     NODE    120
 TERMINAL CELL

 GROUP    3      NO. OF KEYS   529     NO. OF ITEMS    35     NODE    121
 TERMINAL CELL
NO. OF RECORDS ON KEY FILE =    144
NO. OF LEVELS IN TREE =           5
NO. OF RECORDS ON DOCUMENT NODE FILE =  1669
NO. OF NODES IN TREE =          103
```


Figure 3.16



MACRO FLOW CHART FOR STEP 1
PROGRAM NAME: CLASFY

*For each run, the number of these temporary files equals the stratification number used on an input parameter

STEP 1 Quick Reference Chart

Program Name: CLASFY

Function	Section	Name (if any)	Reference in Manual
Input	3.2.3	1. Document Surrogate File (CLSINP)	Table 3.4 pg. 156
Output	3.2.4	1. Document Node File (DOCNDE)	Table 3.6 pg. 164
Input Parameters	3.2.7		Table 3.7 pg. 169
Output Report	3.2.6		Fig. 3.8 pg. 167
Intermediate Files	3.2.5	1. Intermediate Key Files (TKYWD) 2. Intermediate Document Files (TDOC) 3. Redundant Document Files (RESIDOC)	

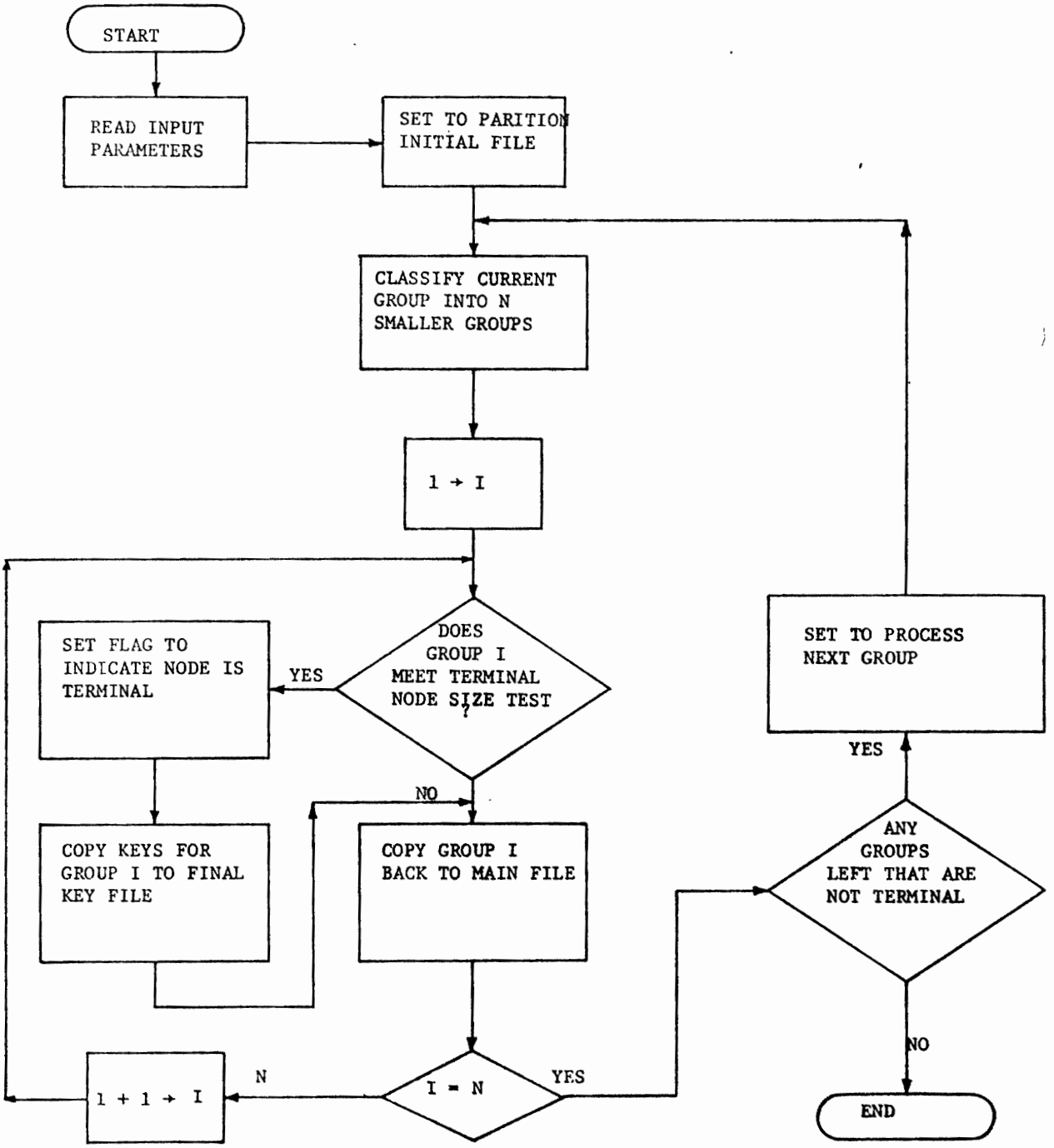
*Names in parentheses indicate names used in Job Control Language.

```
/LOGON LANG,D1035,TIME=20000,PRIORITY=7
/FILE FKYWRD,LINK=DSET22,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN
/FILE DOCNDE,LINK=DSET20,RECFORM=F,RECSIZE=20,FCBTYPE=ISAM,OPEN=OUTIN
/FILE CLSINP,LINK=DSET21,RECFORM=F,RECSIZE=528,FCBTYPE=ISAM,OPEN=INOUT
/FILE RESIDOC,LINK=DSET23,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN
/FILE TKYWD1,LINK=DSET24,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN
/FILE TKYWD2,LINK=DSET25,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN
/FILE TKYWD3,LINK=DSET26,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN
/FILE TDOC1,LINK=DSET27,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN
/FILE TDOC2,LINK=DSET28,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN
/FILE TDOC3,LINK=DSET29,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN
/EXEC LMCLASFR
*03 0075 00050 01000 01000
/LOGOFF                                                    09/05/72
```

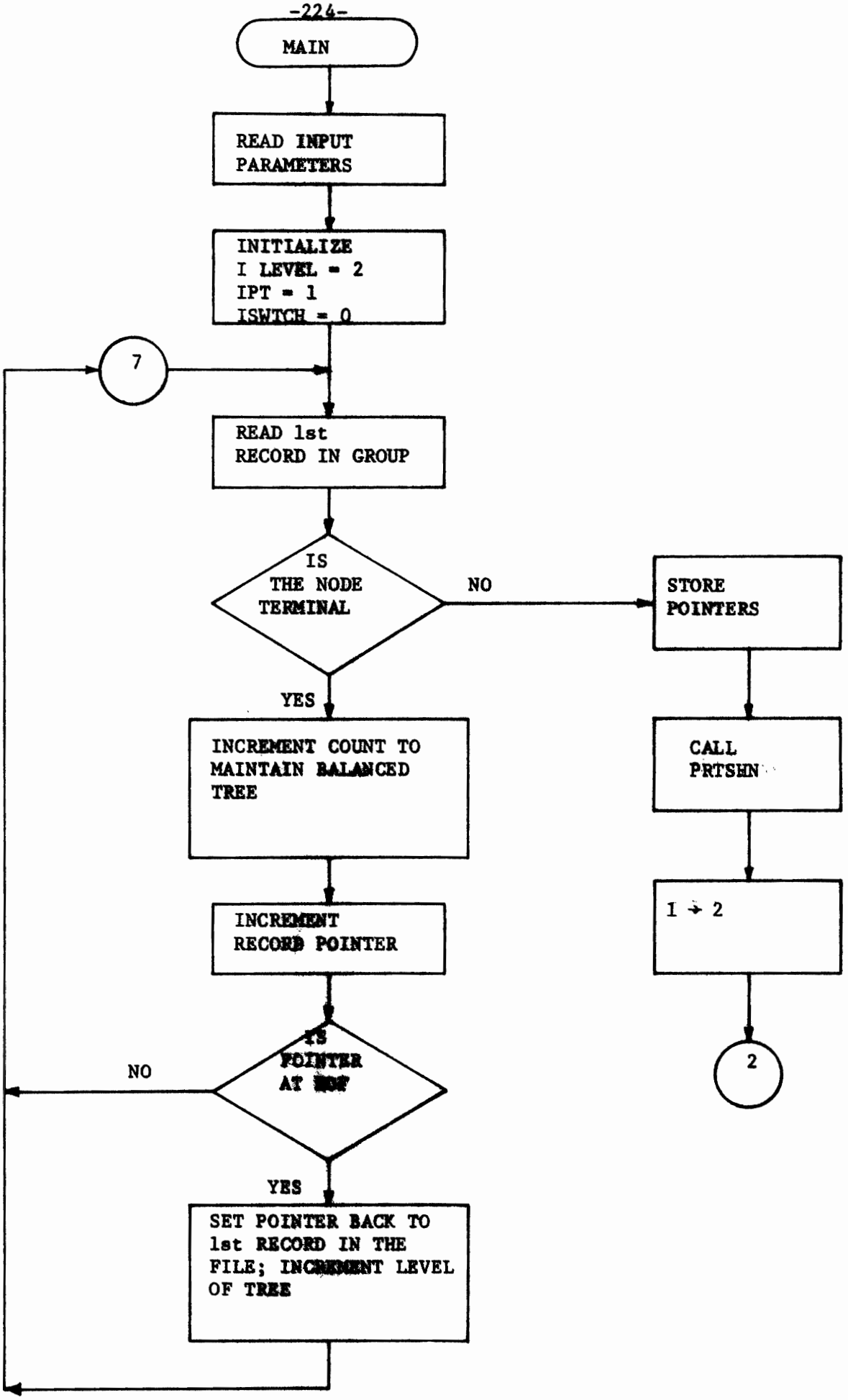
*Input Parameter Card

Job Control Language to Execute Step 1

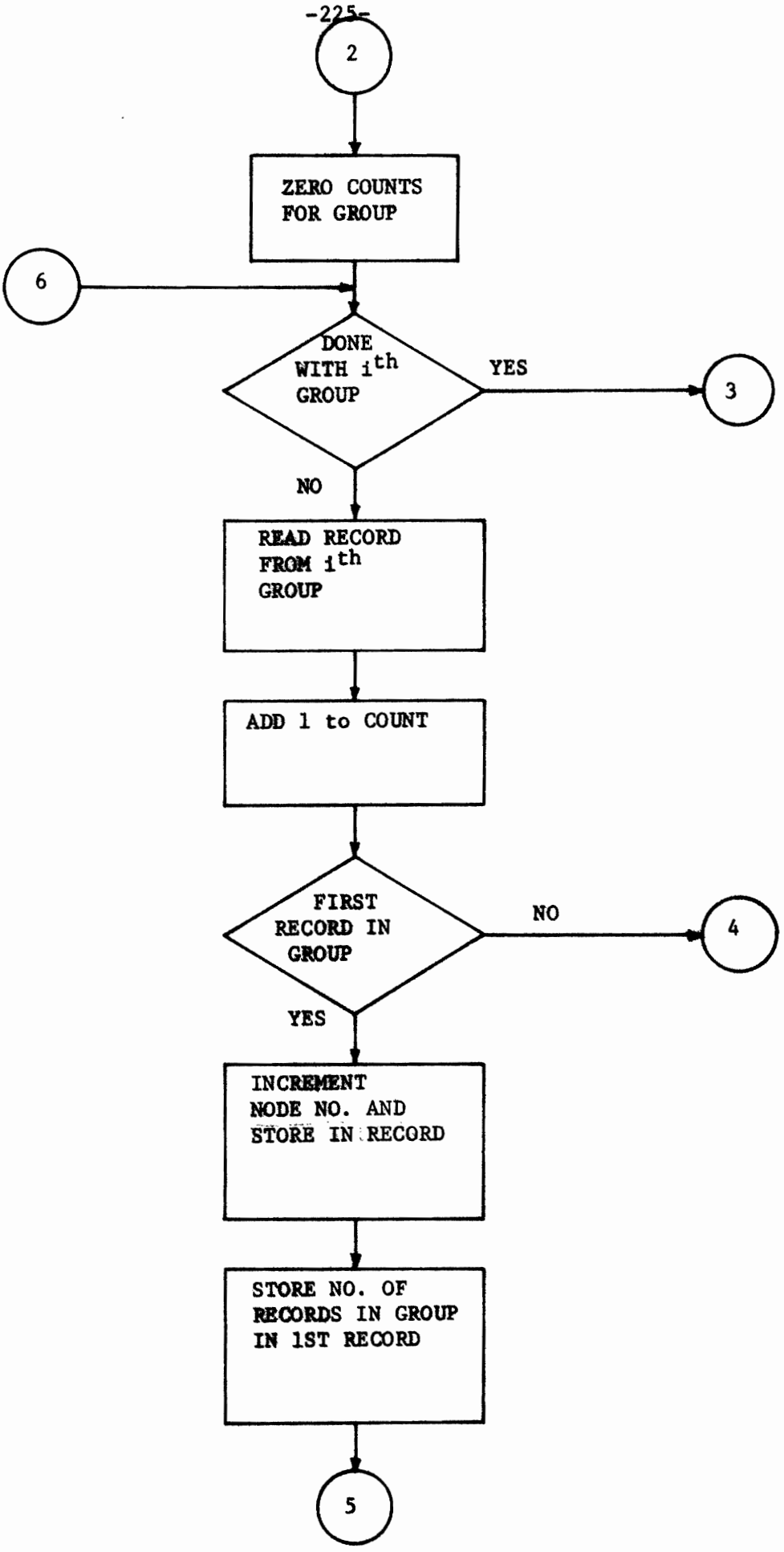
Program Name: CLASFY



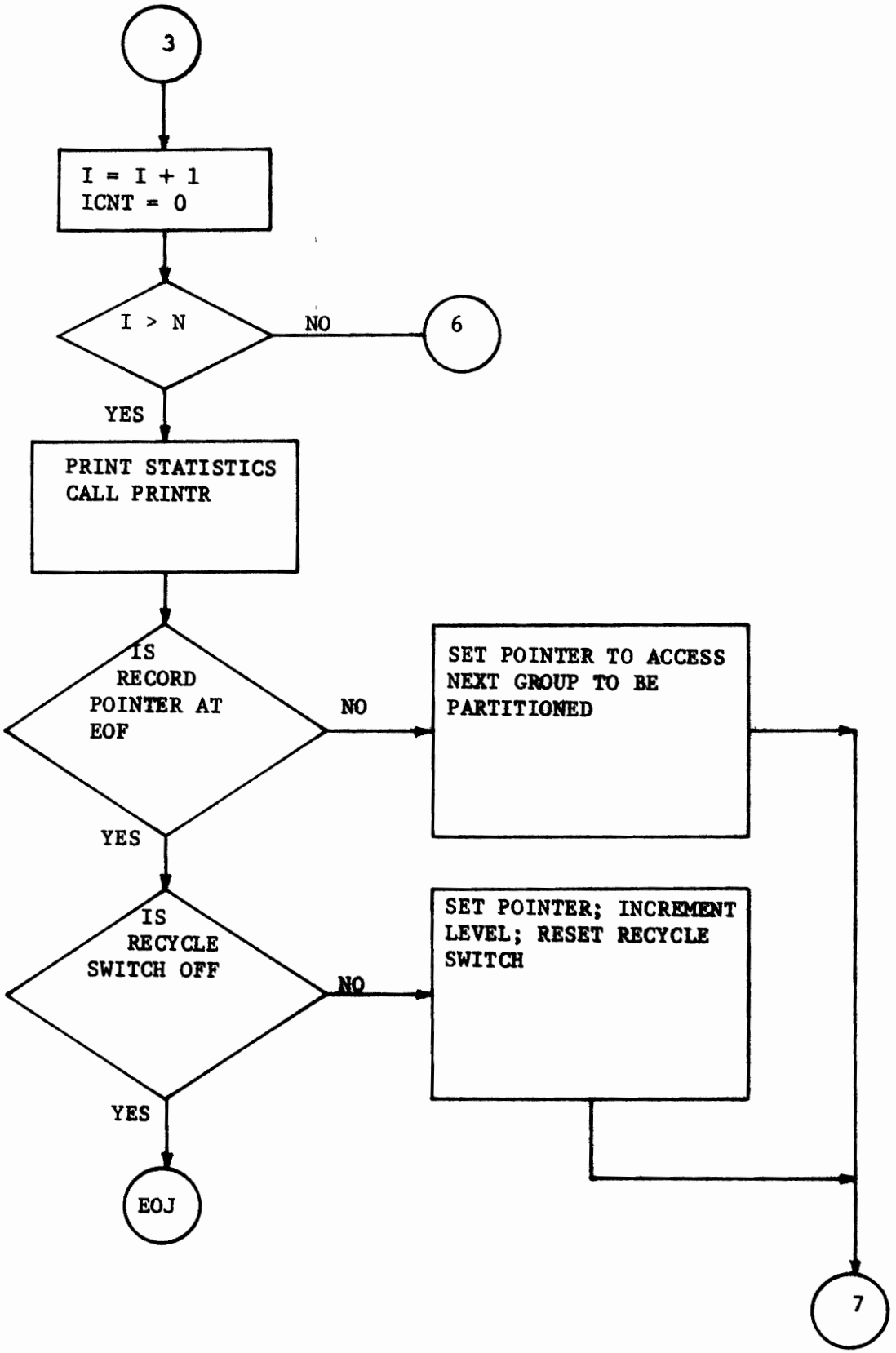
MACRO FLOW CHART FOR CLASFY



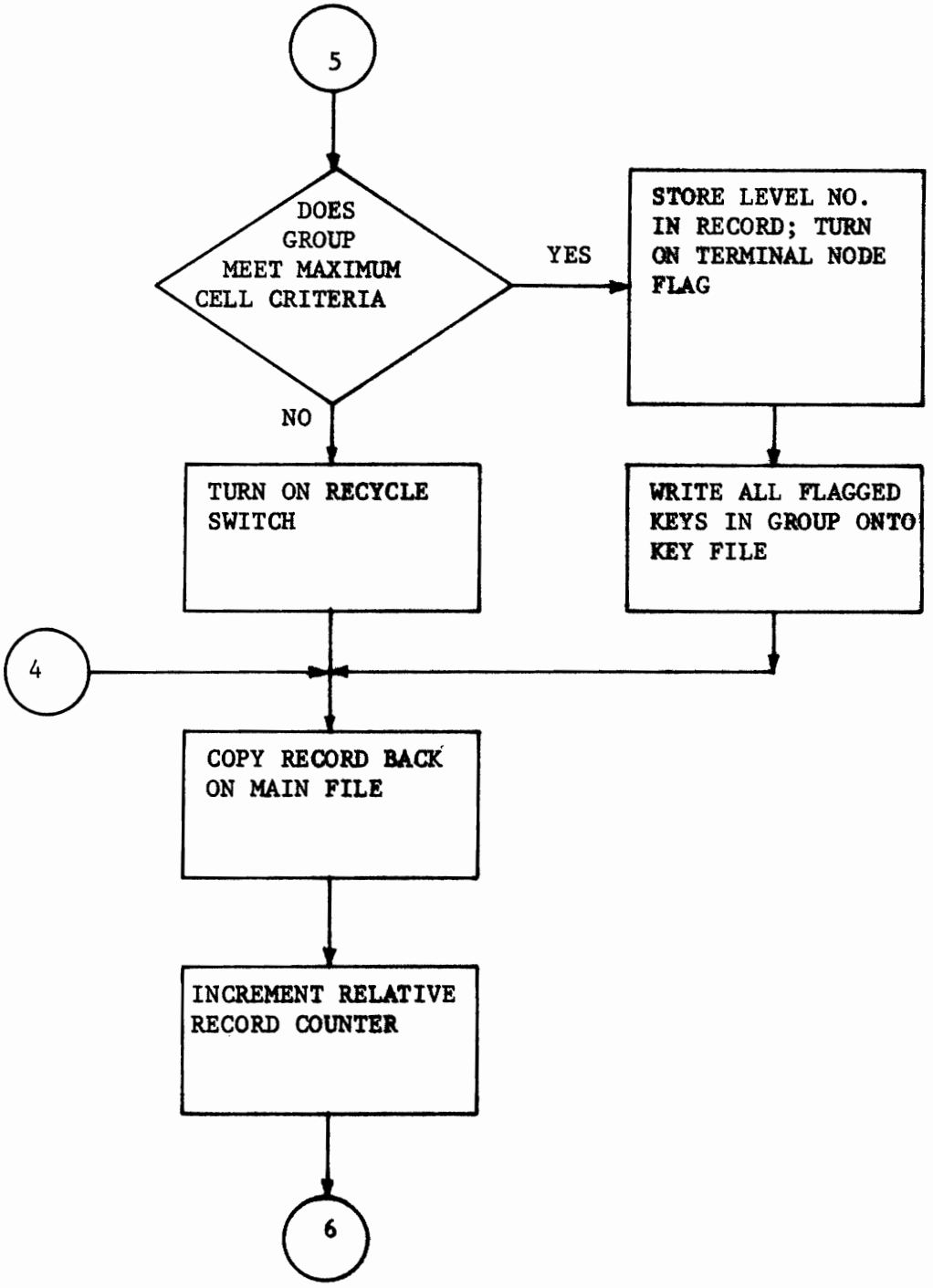
DETAIL FLOW CHART FOR CLASFY



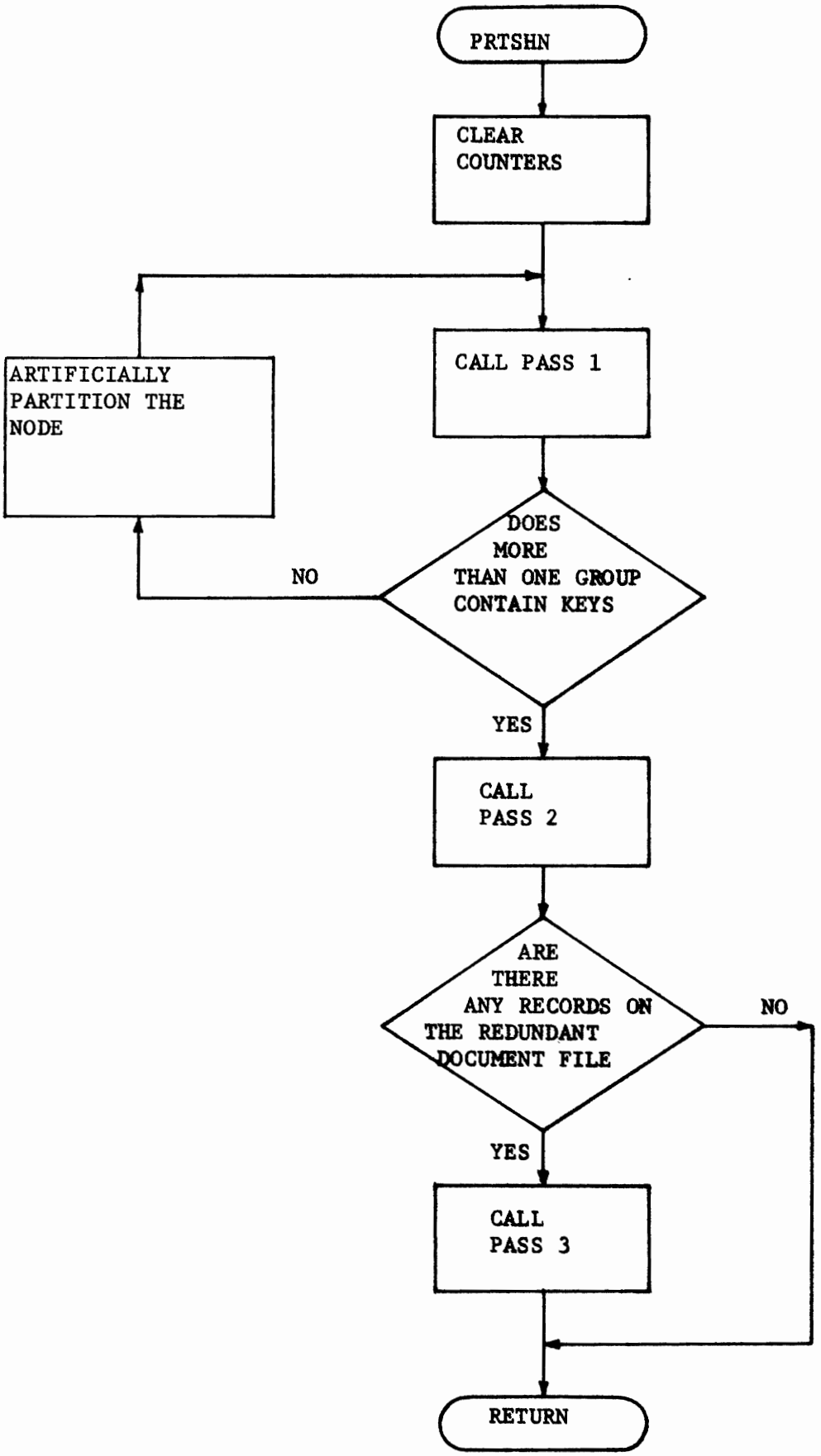
DETAIL FLOWCHART FOR CLASFY



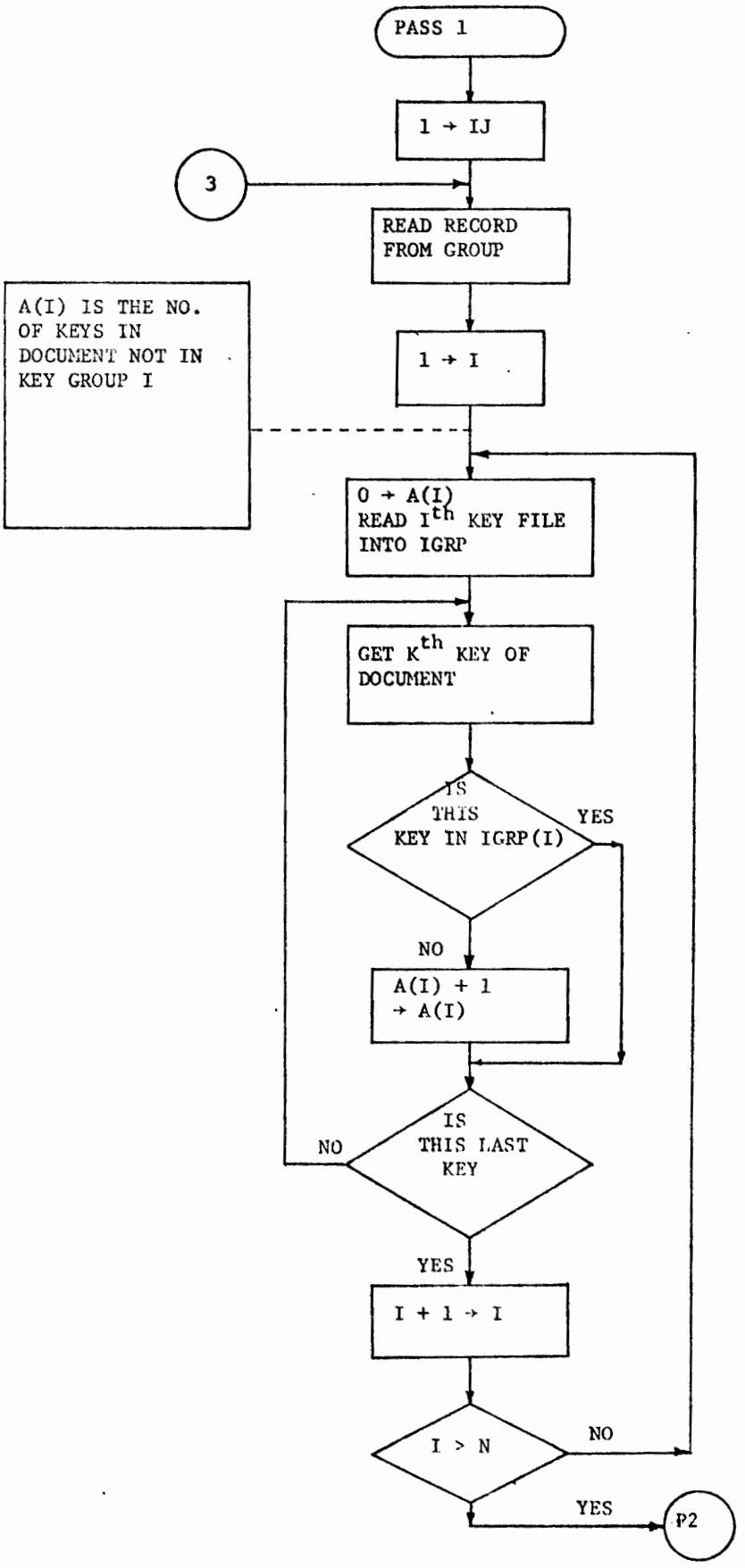
DETAIL FLOWCHART FOR CLASFY



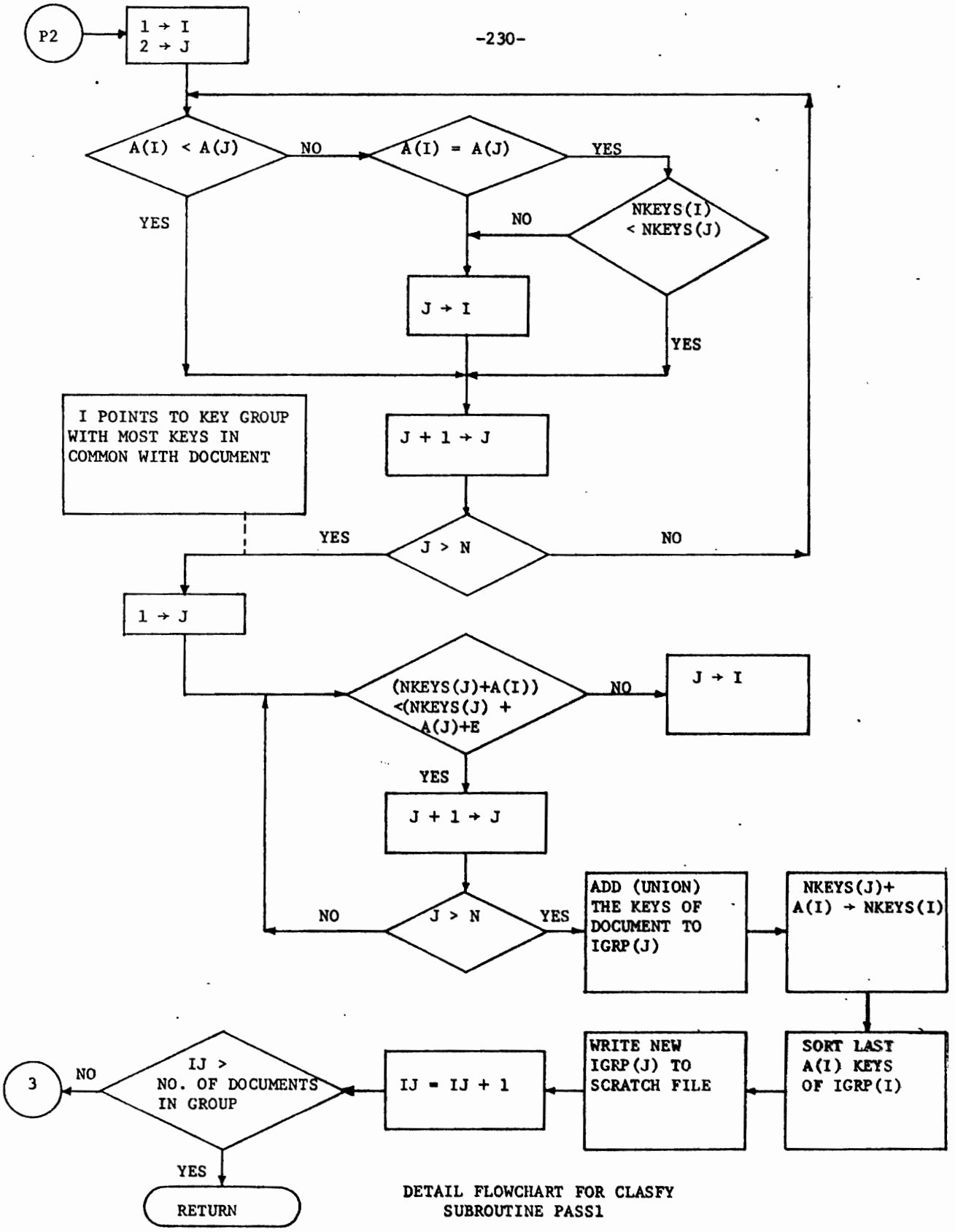
DETAIL FLOW CHART FOR CLASFY



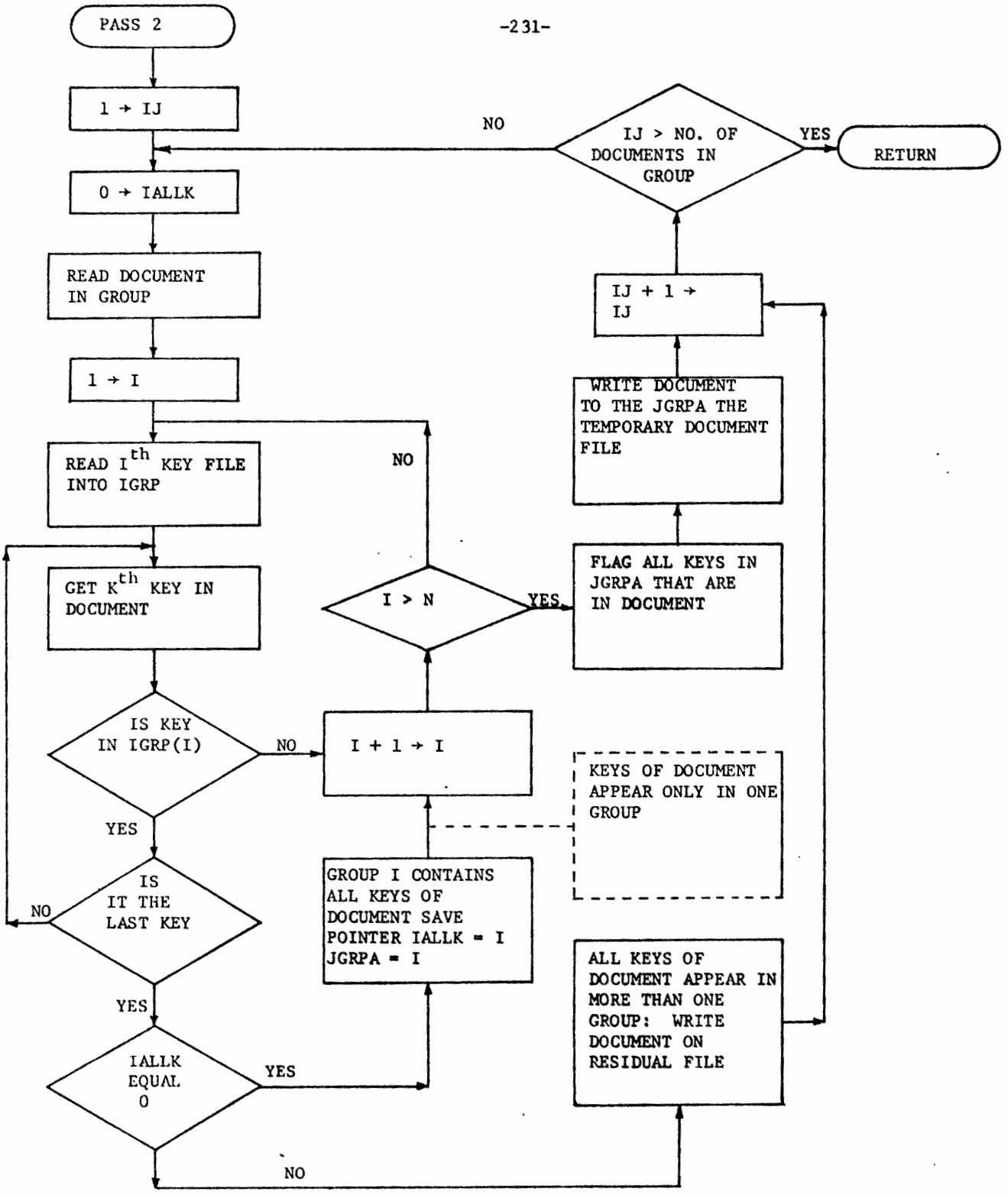
DETAIL FLOWCHART FOR CLASFY SUBROUTINE PRTSHN



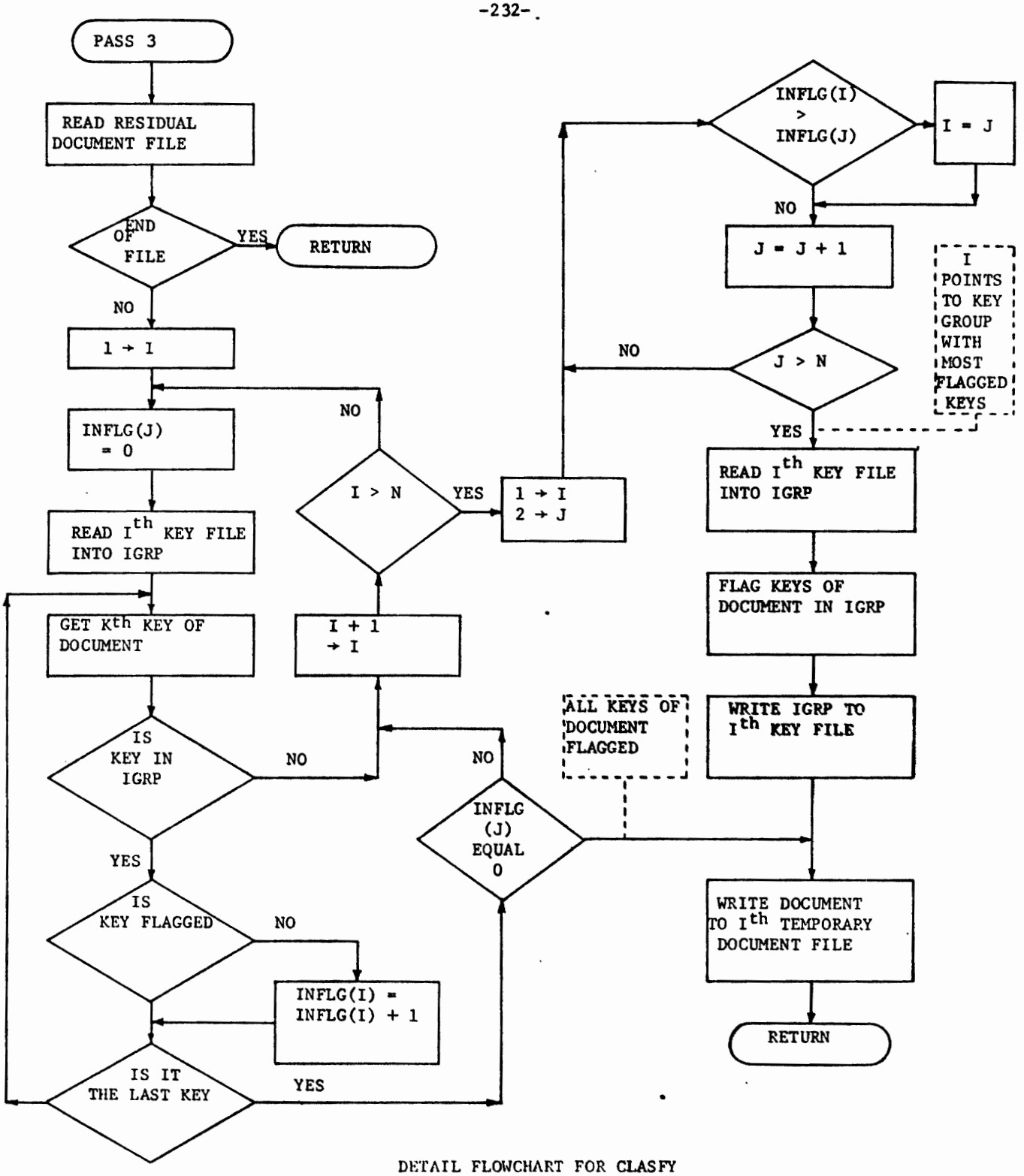
DETAIL FLOWCHART FOR CLASFY SUBROUTINE PASS 1



DETAIL FLOWCHART FOR CLASFY
SUBROUTINE PASS1



DETAIL FLOWCHART FOR CLASFY SUBROUTINE PASS2



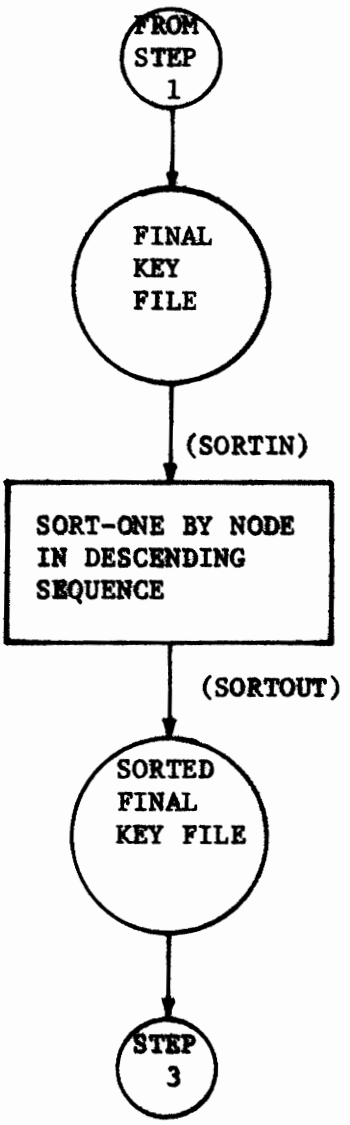
DETAIL FLOWCHART FOR CLASFY
SUBROUTINE PASS 3

3.11.2 STEP 2 Sort One

PROGRAM NAME: SORT

Objective - To sort the final keyword file in descending node number order

The 144 records on the final keyword file were sorted. The sorting was done using the software on the RCA Spectra 70/46.



MACRO FLOW CHART FOR STEP 2
PROGRAM NAME: SORT-ONE

STEP 2 Quick Reference Chart

Program Name: SORT - ONE

Function	Section	Name (if any)	Reference in Manual
Input	3.3	Final Key File (FKYWRD)	Table 3.5 pg. 162
Output	3.3	Sorted Final Key File (SFKYWD)	Table 3.5 pg. 162

*Names in parentheses indicate names used in Job Control Language.

```
/LOGON LANG,D1035,TIME=3000,PRIORITY=7

/ERASE FKYWRD

/STEP

/FILE FKYWRD,LINK=SORTIN,RECFORM=V,BLKSIZE=4096,FCBTYPE=BTAM,          -
/DEVICE=T9N,VOLUME=FKYWRD,OPEN=INPUT,STATE=FOREIGN

/ERASE SFKYWD

/STEP

/FILE SFKYWD,LINK=SORTOUT,RECFORM=V,BLKSIZE=4096,FCBTYPE=BTAM,         -
/DEVICE=T9N,VOLUME=SFKYWD,OPEN=OUTPUT

/EXEC SORT

SORT FIELDS=(13,2,D,15,2,A),FORMAT=FI

RECORD LENGTH=(2034),TYPE=V

END

/LOGOFF
```

Job Control Language to Execute Step 2

Program Name: Sort-One

3.11.3 STEP 3 Build Classification Tree

PROGRAM NAME: TREE

Objective: To build a classification tree (from the Final Keyword File) to be used in retrievals and browsing.

The 69 terminal nodes on the final keyword file were used to create the final classification tree. Table 3.18 indicates each node of the classification tree along with the number of keys in that node. (This table is a tabulation of the number of keys found at each node in the Node-To-Key table.) The shape of the classification tree is the same as the intermediate classification tree shown in Figure 3.15. Only the keys associated with each node in the tree has changed. Figure 3.17 shows the control totals from the TREE program.

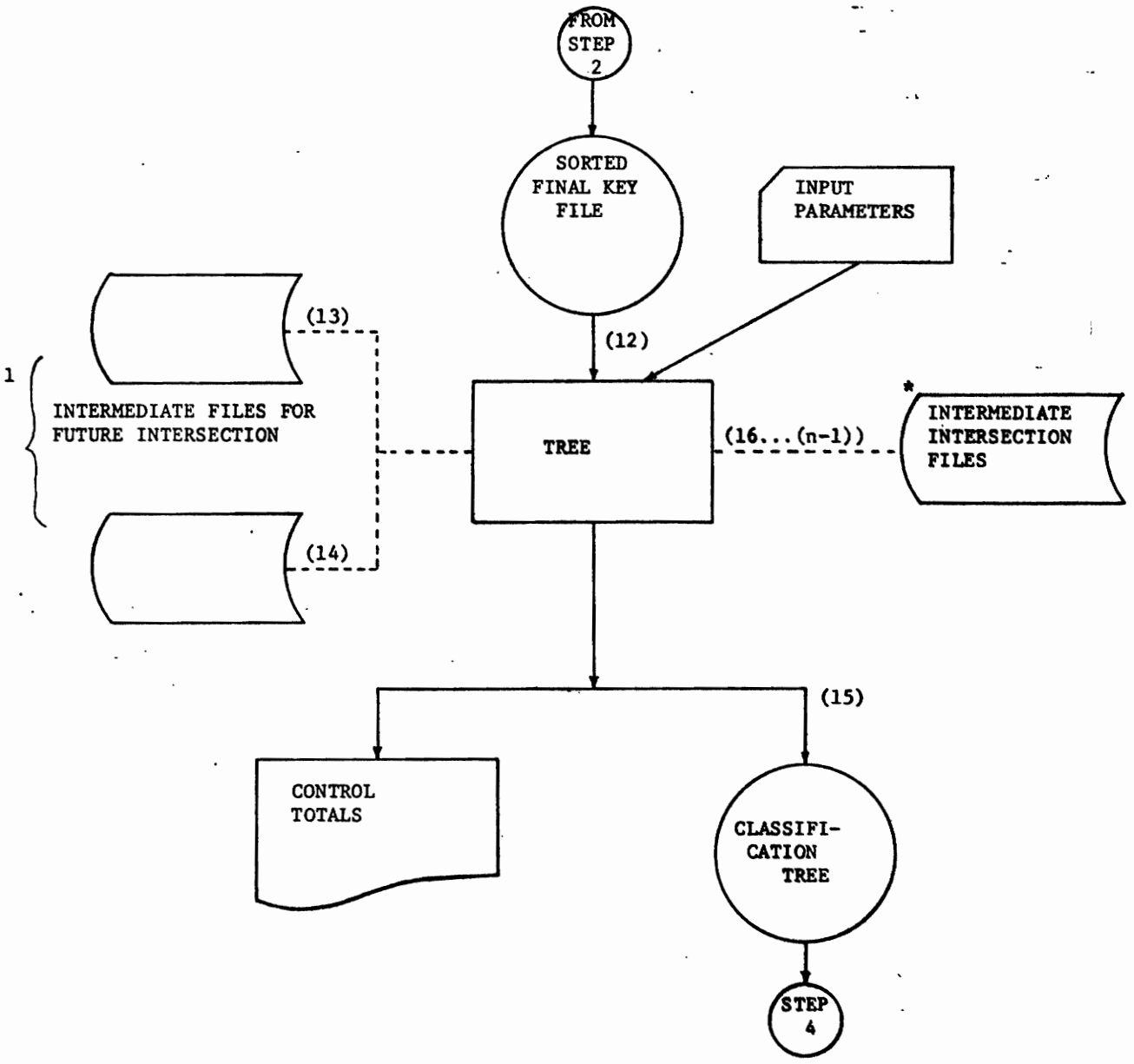
Node	Number of Keys	Node	Number of Keys	Node	Number of Keys	Node	Number of Keys
1	4	27	130	53	377	82	464
2	14	28	96	54	326	83	464
3	12	29	123	55	409	84	460
4	25	30	156	56	413	85	458
5	25	31	71	57	333	86	395
6	38	32	669	58	415	87	446
7	32	33	709	59	382	88	386
8	116	34	636	60	416	89	450
9	50	35	1063	61	365	90	453
10	20	36	91	62	392	91	391
11	419	37	53	63	389	92	441
12	22	38	1056	64	328	93	358
13	32	39	107	65	317	94	451
14	88	40	59	66	368	107	444
15	79	41	428	67	403	108	384
16	136	42	360	71	462	109	442
17	44	43	360	72	390	110	389
18	123	44	422	73	383	111	362
19	163	45	424	74	452	112	441
20	117	46	333	75	419	116	429
21	66	47	405	76	492	117	473
22	106	48	382	77	471	118	418
23	1078	49	339	78	477	119	399
24	151	50	374	79	441	120	468
25	109	51	357	80	463	121	409
26	187	52	433	81	427		

TABLE 3.18

Keys in Final Classification Tree

```
THE FOLLOWING IS THE INPUT PARAMTER CARD
  3   5    144  1000  1000
NO. OF RECORDS ON TREE FILE=    209
NO. OF NODES ON TREE FILE=    103
NO. OF KEYS IN TREE COUNTING DUPLICATES =  34077
```


Figure 3.17



MACRO FLOW CHART FOR STEP 3
Program Name: TREE

*For each run the number of these temporary files equals the stratification number for the tree

STEP 3 Quick Reference Chart

Program Name: TREE

Function	Section	Name (if any)	Reference in Manual
Input	3.4.2	Sorted Final Key File (SKKYWD)	Table 3.5 pg. 162
Output	3.4.4	Classification Tree (CLTREE)	Table 3.9 pg. 186
Input Parameters	3.4.5		Table 3.10 pg. 187
Intermediate Files	3.4.3	Intermediate Intersection Files (SCR)	
		Intermediate Files for Future Intersection (TEMP1, TEMP2)	

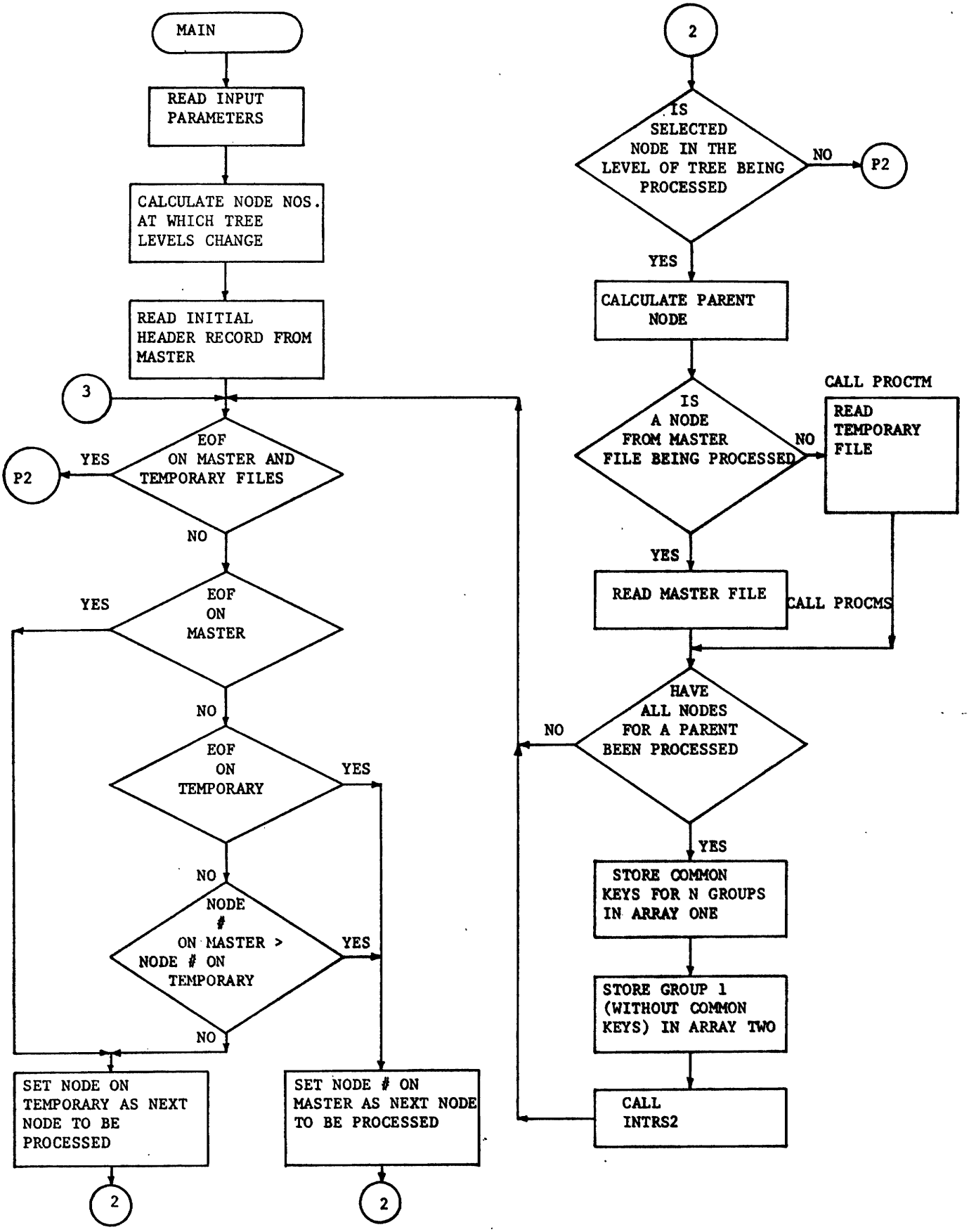
*Names in parentheses indicate names used in Job Control Language.

```
/LOGON LANG,D1035,TIME=5000,PRIORITY=7

/ERASE SFKYWD

/STEP

/FILE SFKYWD,LINK=DSET12,RECFORM=V,BLKSIZE=4096,FCBTYPE=BTAM,          -
/DEVICE=T9N,VOLUME=SFKYWD,STATE=FOREIGN,OPEN=INPUT

/FILE TEMP1,LINK=DSET13,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN

/FILE TEMP2,LINK=DSET14,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN

/FILE SCR1,LINK=DSET16,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN

/FILE SCR2,LINK=DSET17,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN

/FILE SCR3,LINK=DSET18,RECFORM=V,BLKSIZE=STD,FCBTYPE=ISAM,OPEN=OUTIN

/FILE CLTREE,LINK=DSET15,RECFORM=V,BLKSIZE=4096,FCBTYPE=BTAM,          -
/DEVICE=T9N,VOLUME=CLTREE,OPEN=OUTPUT

/EXEC LMTREE

*003 005 000144 01000 01000

/LOGOFF
```

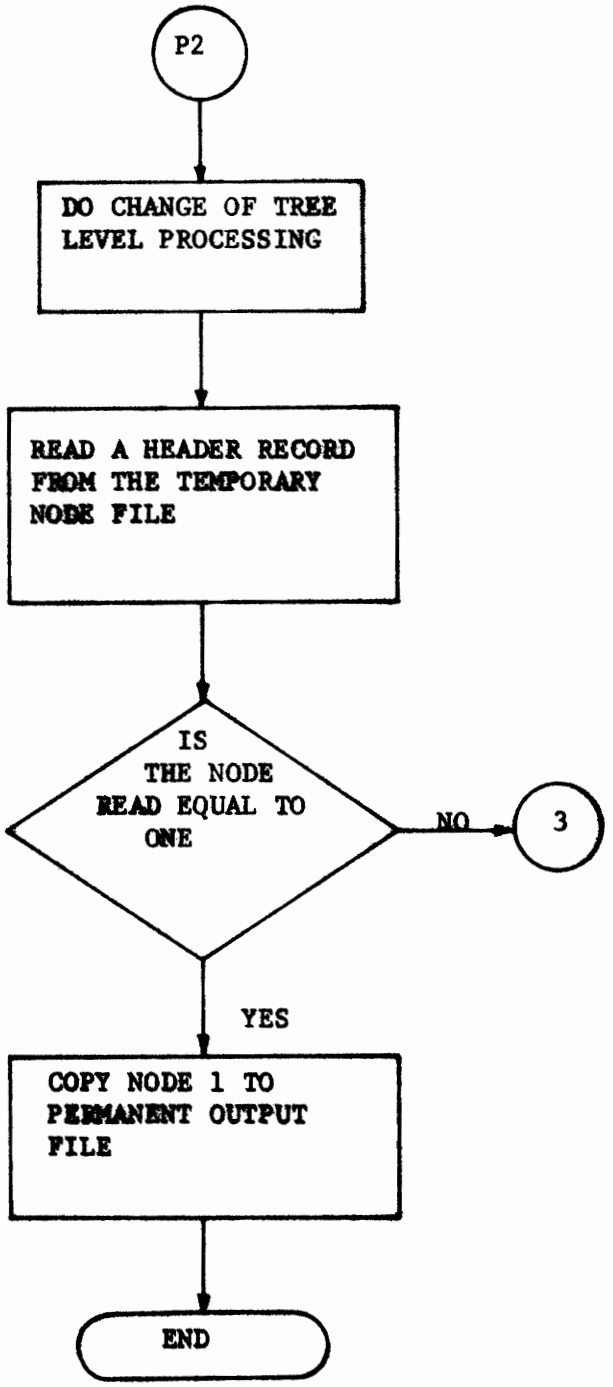
*Input Parameter Card

Job Control Language to Execute Step 3

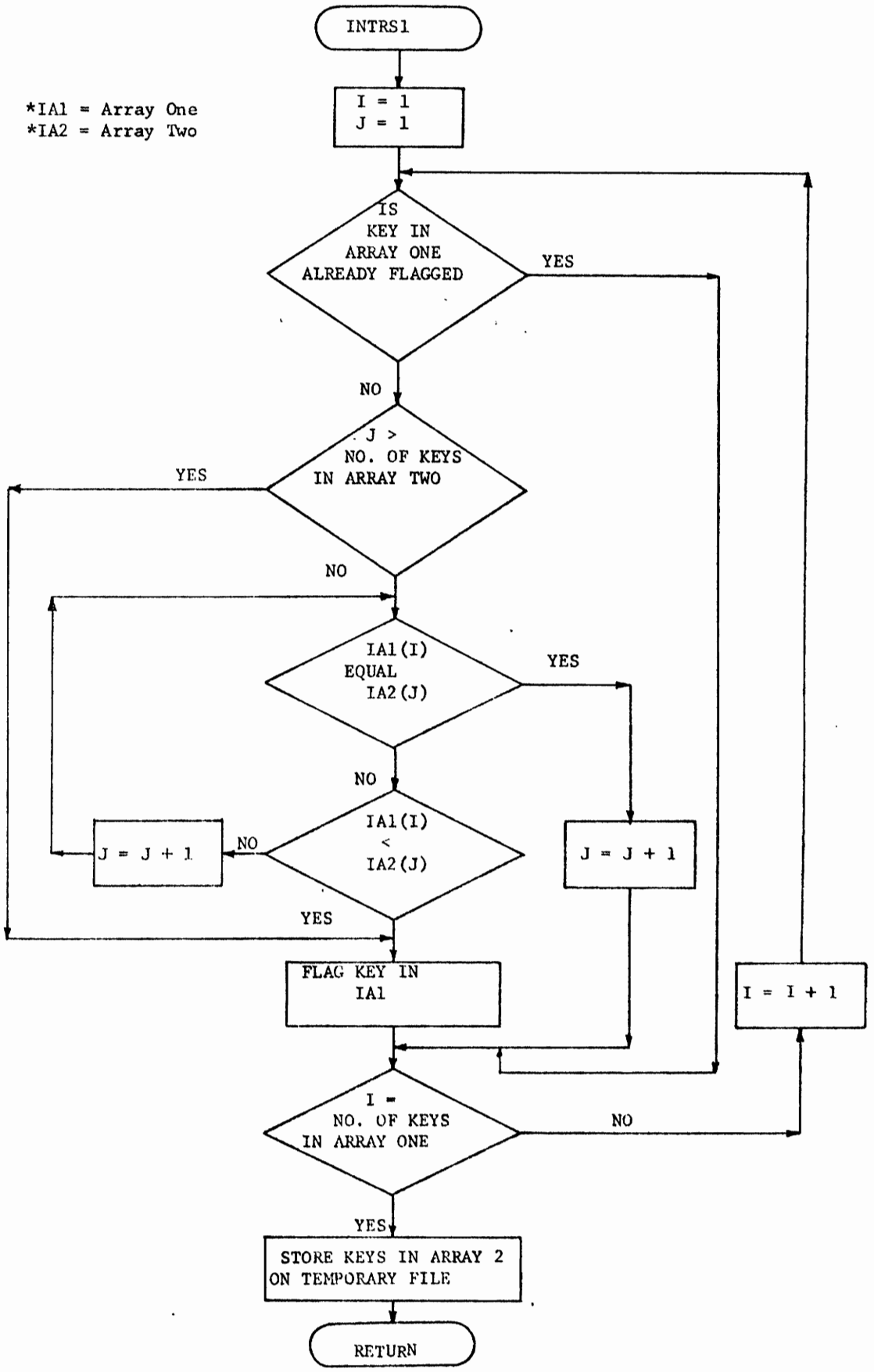
Program Name: Tree



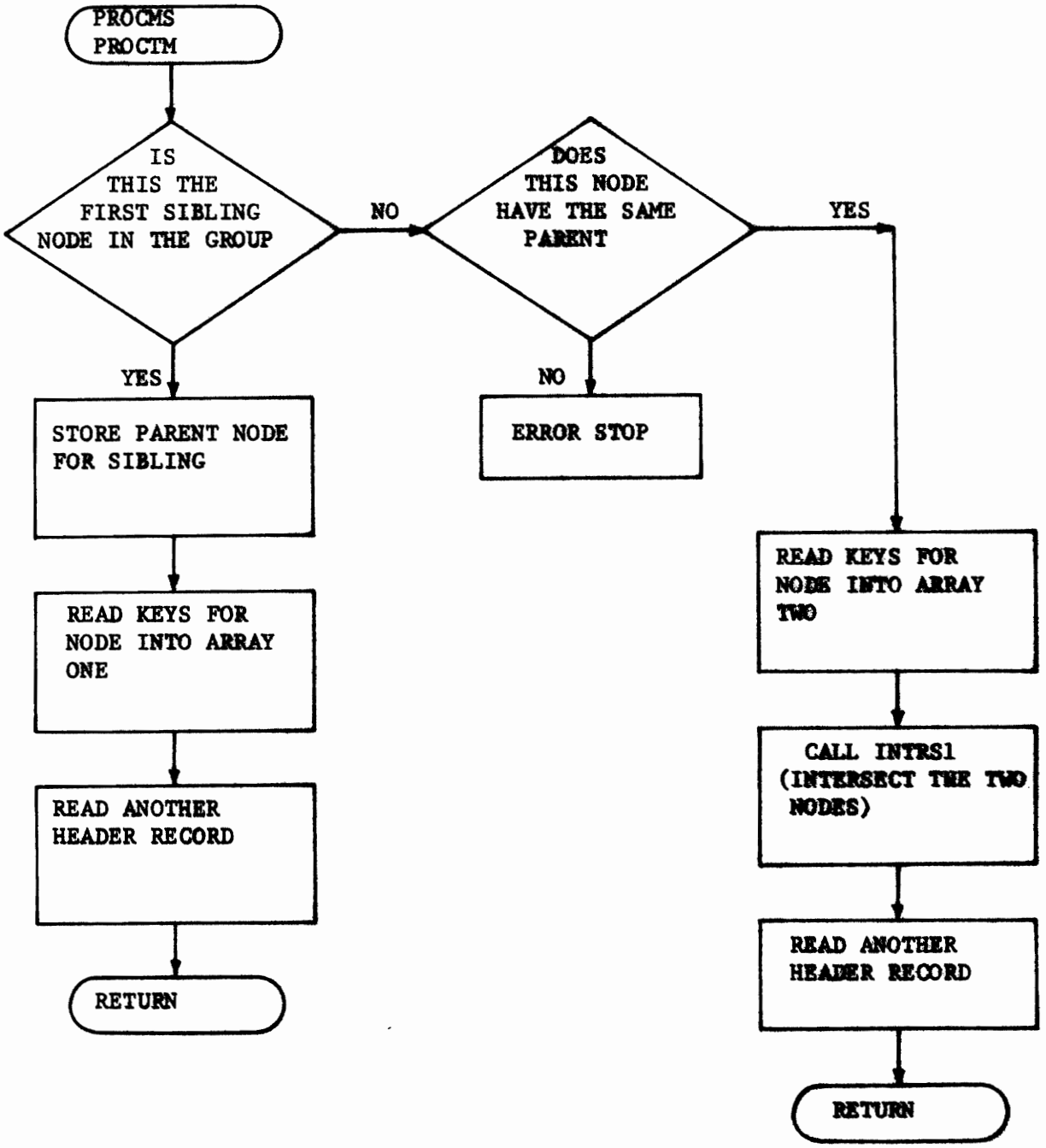
DETAIL FLOWCHART FOR TREE



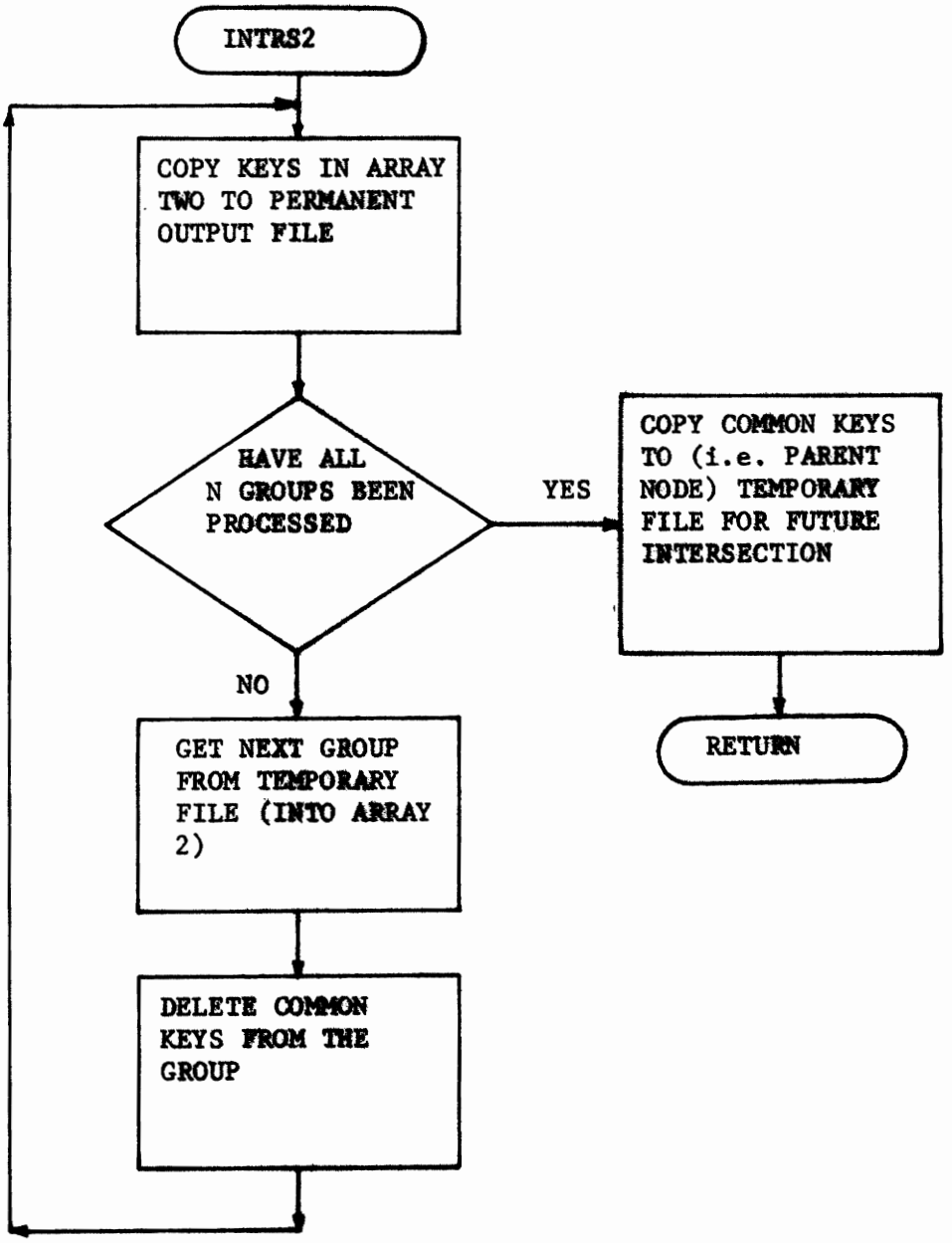
DETAIL FLOW CHART FOR TREE



DETAIL FLOWCHART FOR TREE SUBROUTINE INTRS1



DETAIL FLOWCHART FOR TREE
SUBROUTINES PROCMS AND PROCTM



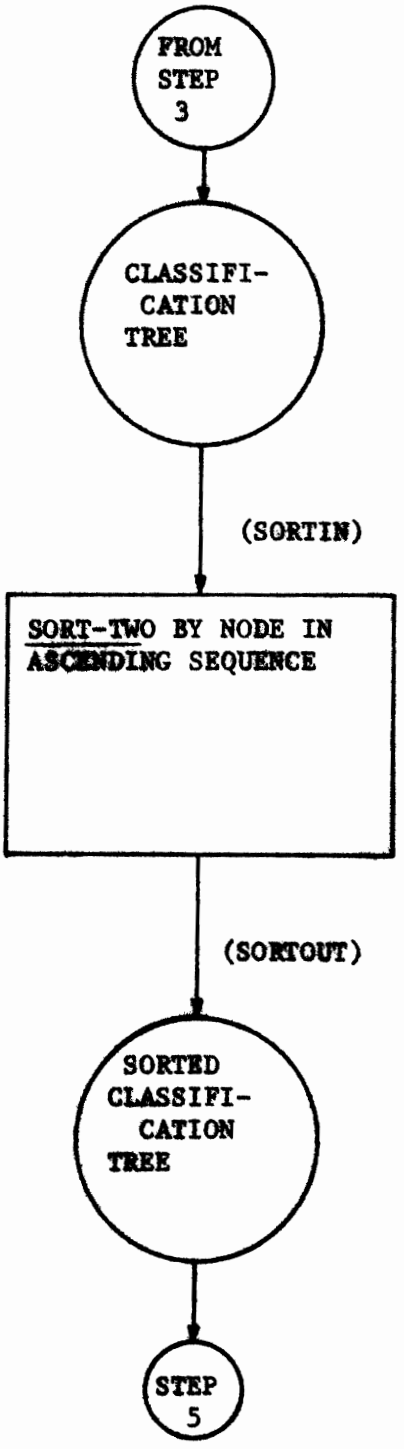
DETAIL FLOWCHART FOR TREE
SUBROUTINE INTRS2

3.11.4 STEP 4 Sort Two

PROGRAM NAME: SORT

OBJECTIVE: To sort The Classification Tree in ascending node number order

The classification tree was sorted is ascending node number order. Again, the RCA SPECTRA system sort package was used.



MACRO FLOW CHART FOR STEP 4
PROGRAM NAME: SORT-TWO

STEP 4 Quick Reference Chart

Program Name: SORT - TWO

Function	Section	Name (if any)	Reference in Manual
Input	3.5	Classification Tree (CLTREE)	Table 3.9 pg. 186
Output	3.5	Sorted Classification Tree (SCLTRE)	Table 3.9 pg. 186

*Names in parentheses indicate names used in Job Control Language.

```
/LOGON LANG,D1035,TIME=3000,PRIORITY=7

/ERASE CLTREE

/STEP

/FILE CLTREE,LINK=SORTIN,RECFORM=V,BLKSIZE=4096,FCBTYPE=BTAM,          -
/DEVICE=T9N,VOLUME=CLTREE,OPEN=INPUT,STATE=FOREIGN

/FILE SCLTRE,LINK=SORTOUT,RECFORM=V,BLKSIZE=4096,FCBTYPE=BTAM,         -
/DEVICE=T9N,VOLUME=SCLTRE,OPEN=OUTPUT

/EXEC SORT

SORT FIELDS=(13,2,A,15,2,A),FORMAT=FI

RECORD LENGTH=(2034),TYPE=V

END

/LOGOFF
```

Job Control Language to Execute Step 4

Program Name: Sort-Two

3.11.5 STEP 5 Node-To-Key

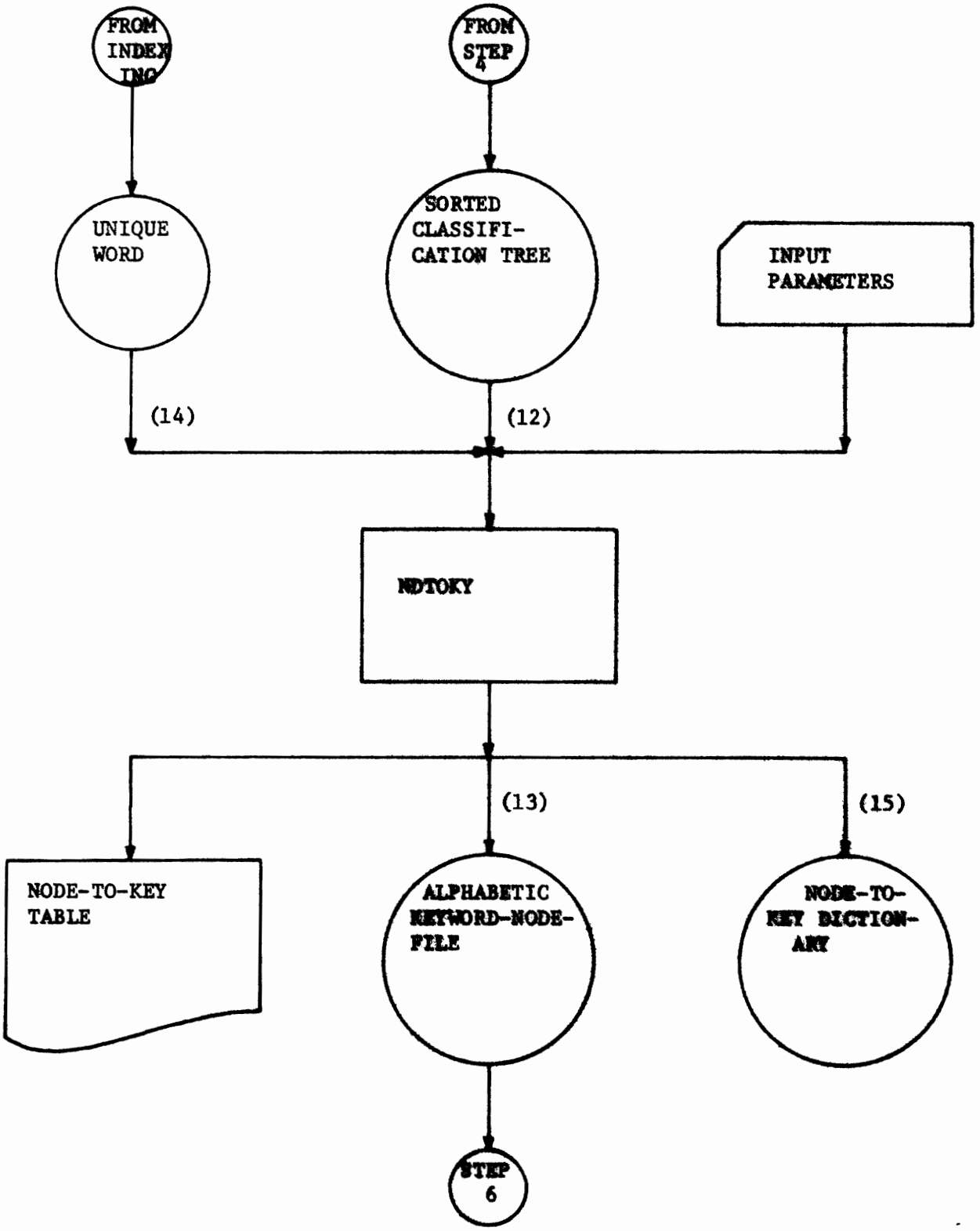
PROGRAM NAME: NDTOKY

OBJECTIVE: To create the Node-To-Key dictionary which is of direct use in browsing

The node-to-key dictionary was created for the experimental data base. A sample of the Node-To-Key table is shown in Figure 3.18. The number of nodes in this table is 103, which is equivalent to the number of nodes in the classification tree.

NODE	KEYS
1	APRIL FORCE INTERNATIONAL LEAD
1,1	ARM COUNTRY DEVELOP ENGLISH FOREIGN FRIEND GENERAL GOVERNMENT MEET MILITARY PRESENT REPORT UNITE VISIT
1,2	ARMY ARM DEFENSE DOMESTIC MARCH MILITARY NATIONAL REVOLUTION STATE SUPPORT US WAR
1,3	COMMITTEE COUNTRY CREATE DEVELOP DOMESTIC ENGLISH FOREIGN GENERAL GOVERNMENT MARCH MEET MEMBER MINISTER NATIONAL OFFICIAL PEACE PLAN POLITICAL PRESENT PUBLIC RADIO REPORT SECURITY STATE UNITE
1,1,1	AID ARMY ARRIVE BORDER DEFENSE DOMESTIC EAST INDIA ISRAEL KARACHI LAW MEMBER OFFICE OFFICIAL PAKISTAN POS PEACE POWER RETURN SECURITY SOVIET STATE SUPPORT USSR WEST
1,1,2	AFFAIR AGGRESIVE AIR CHAIRMAN CHIEF CHINA COMMUNIST COMRADE CONGRESS COUNCIL DEMOCRATIC DEPARTMENT HEAD HONOR HSINHUA LIBERATE MAO MARCH MINISTER NATIONAL NCNA PARTY PEKING POLITICAL PRESIDENT RELATION REVOLUTION SECOND SOCIALIST SOUTH SPEECH STATE TALK UNION US VICTORY VIETNAM WAR
1,1,3	AMBASSADOR COMMUNIST CONFERENCE CONGRESS COOPERATE CREATE DELEGATE DOMESTIC EAST ECONOMIC MEMBER MINISTER MOSCOW NATIONAL PARTY PEACE PLAN POLICY POLITICAL POWER PRESIDENT PRINCIPLE RELATION RESOLUTION RUSSIA SOVIET SPEECH SUPPORT TALK VIEW WAR ZONE
1,2,1	ADMINISTRATE ADMIT ADOPT ADVANCE ADVENTURE AGGRESIVE AGGRESSIVE AMERICAN ANNIHILATE ANSWER ARTILLERY ASIA ATTACK ATTEMPT BANKRUPT BASE BATTLE BOMB CAMBODIA CAPITALIST CAUSE CENTRAL CLIQUE COMBAT COMMITTEE COMMUNIST COMPATRIOT CONFERENCE COUNTRY COURSE DECISION DEFEAT DEMOCRATIC DESTROY DEVELOP DISASTER DOCTRINE ENEMY ENGLISH ESCALATE EXPAND EXPLOIT FAIL FIGHT FIRE FORFIGN FRATERNAL FREEDOM FRIEND FRONT GOVERNMENT HEAD HOME IMPERIALIST INDEPENDENT INDOCHINA INVOLVE ISOLATE JOINT LAOS LAUNCH LIBERATE LOCAL LOSS MANEUVER MEET NAME NAM NATURE NGUYEN NIXON NORTH NUMBER OPERATE PEACE PLAN POLICY POLITICAL POWER PRESENT PRESIDENT PROGRAM PROGRESS PUBLIC PUNISH PUPPET REACTION REASON REPEAT REPUBLIC RESISTANCE RESULT RULE SACRIFICE SAIGON SETBACK SOLDIER SOUTH SPRING STRATEGY STRENGTH STRONG STRUGGLE TALK TEN TERRITORY THOUSAND THU TRICK TROOP UNCONDITIONAL UNITE VICTORY VIETNAMIZATION VIETNAM ZONE
1,2,2	AGGRESIVE ARABIC ARAB BATTLE CENTRAL COMMITTEE COUNTRY DEFEND DEMOCRATIC DEVELOP ECONOMIC EGYPT ESTABLISH FIGHT FOREIGN FRATERNAL FRIEND FRONT GENERAL GOVERNMENT HISTORY

Figure 3.18



MACRO FLOW CHART FOR STEP 5
PROGRAM NAME: NOTOKY

STEP 5 Quick Reference Chart

Program Name: NDTOKY

Function	Section	Name (if any)	Reference in Manual
Input	3.6.3	1. Sorted Classification Tree (SCLTRE)	Table 9 pg. 186
		2. Unique-Word/Code (WRCOD)	Fig. 2.14 pg. 145
Output	3.6.4	1. Alphabetic Key-Node File (KYNDE)	Table 3.11 pg. 195
		2. Node-To-Key Dictionary (NODEKY)	Table 3.12 pg. 197
Input Parameters	3.6.5		Table 3.13 pg. 198
Output Report	3.6.6	Node-To-Key Table	Fig. 3.13 pg. 200

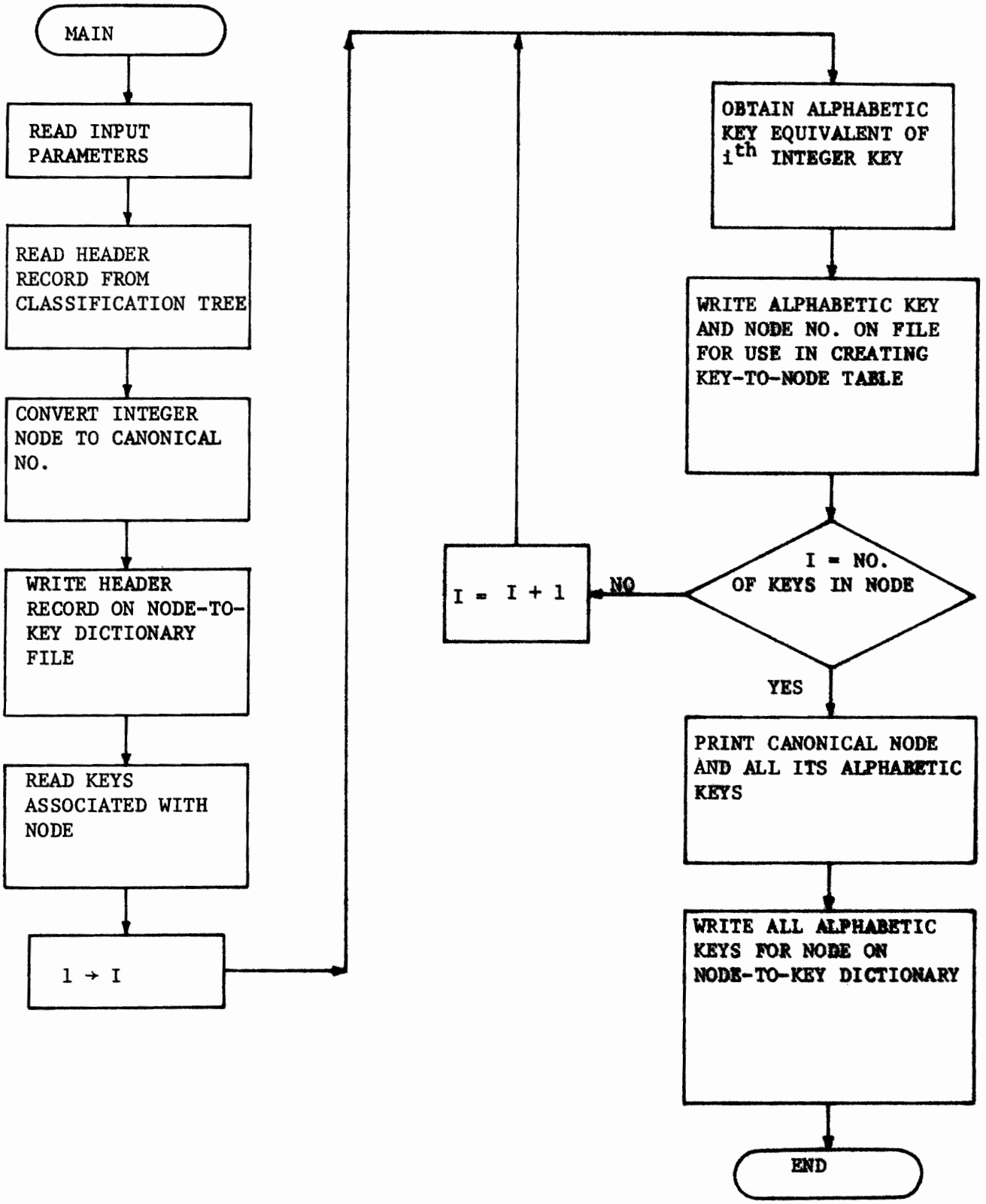
*Names in parentheses indicate names used in Job Control Language.

```
/LOGON LANG,D1035,TIME=10000,PRIORITY=7

/ERASE SCLTRE

/STEP

/FILE SCLTRE,LINK=DSET12,RECFORM=V,BLKSIZE=4096,FCBTYPE=BTAM,                -
/DEVICE=T9N,VOLUME=SCLTRE,OPEN=INPUT,STATE=FOREIGN

/FILE WRDCOD,LINK=DSET14,RECFORM=F,RECSIZE=54,BLKSIZE=STD,                   -
/FCBTYPE=ISAM,OPEN=INPUT

/ERASE NODEKY

/STEP

/FILE NODEKY,LINK=DSET15,RECFORM=V,BLKSIZE=4096,FCBTYPE=BTAM,                -
/DEVICE=T9N,VOLUME=NODEKY,OPEN=OUTPUT

/ERASE KYNDE

/STEP

/FILE KYNDE,LINK=DSET13,RECFORM=F,RECSIZE=46,BLKSIZE=4094,                   -
/FCBTYPE=BTAM,DEVICE=T9N,VOLUME=KYNDE,OPEN=OUTPUT

/EXEC LMNDTOKY

*01000005003020010

/LOGOFF
```

*Input Parameter Card

Job Control Language to Execute Step 5

Program Name: NDTOKY



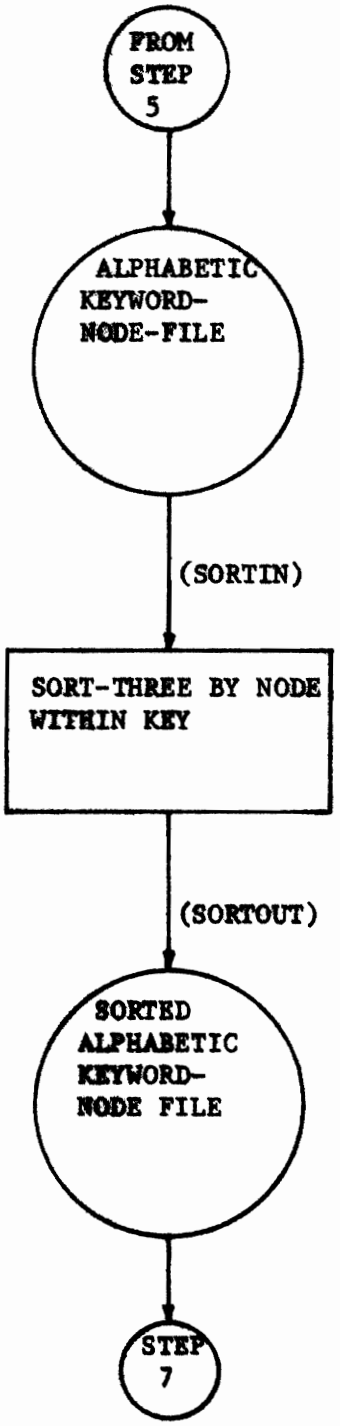
DETAIL FLOWCHART FOR NOTOKY

3.11.6 STEP 6 SORT THREE

PROGRAM NAME: SORT

OBJECTIVE: To sort the alphabetic key-node file in ascending alphabetic key sequence.

The alphabetic key-node file was sorted in ascending alphabetic key sequence using the Spectra System sort. The number of records on this file for the experimental data base was 34077. This number is a count of all keys in the tree, including duplicates.



MACRO FLOW CHART FOR STEP 6
PROGRAM NAME: SORT-THREE

STEP 6 Quick Reference Chart

Program Name: SORT - THREE

Function	Section	Name (if any)	Reference in Manual
Input	3.7	Alphabetic Keyword-Node File (KYNDE)	Table 3.11 pg. 195
Output	3.7	Sorted Alphabetic Keyword-Node File (SKYNDE)	Table 3.11 pg. 195

*Names in parentheses indicate names used in Job Control Language.

```
/LOGON LANG,D1035,TIME=5000,PRIORITY=7

/ERASE KYNDE

/STEP

/FILE KYNDE,LINK=SORTIN,RECFORM+F,RECSIZE=46,BLKSIZE=4094,                -
/FCBTYPE=SAM,DEVICE=T9N,VOLUME=KYNDE,STATE=FOREIGN,OPEN=INPUT

/ERASE SKYNDE

/STEP

/FILE SKYNDE,LINK=SORTOUT,RECFORM=F,RECSIZE=46,BLKSIZE=4094,              -
/FCBTYPE=SAM,DEVICE=T9N,OPEN=OUTPUT

/ERASE QDISK

/STEP

/FILE QDISK,LINK=SORTWK,VOLUME=LANG01,DEVICE=D590,SPACE=(10000,1000)

/EXEC SORT

SORT FIELDS=(7,2,A,9,2,A,11,2,A,13,2,A,15,2,A,a7,2,A,19,2,A,

21,2,A,23,2,A,25,2,A,27,2,A,29,2,A,31,2,A,33,2,A,35,2,A,37,2,A,

39,2,A,41,2,A,43,2,A,45,2,A,5,2,A),FORMAT=FI

END

/LOGOFF
```

Job Control Language to Execute Step 6

Program Name: Sort-Three

3.11.7 STEP 7 KEY-TO-NODE TABLE

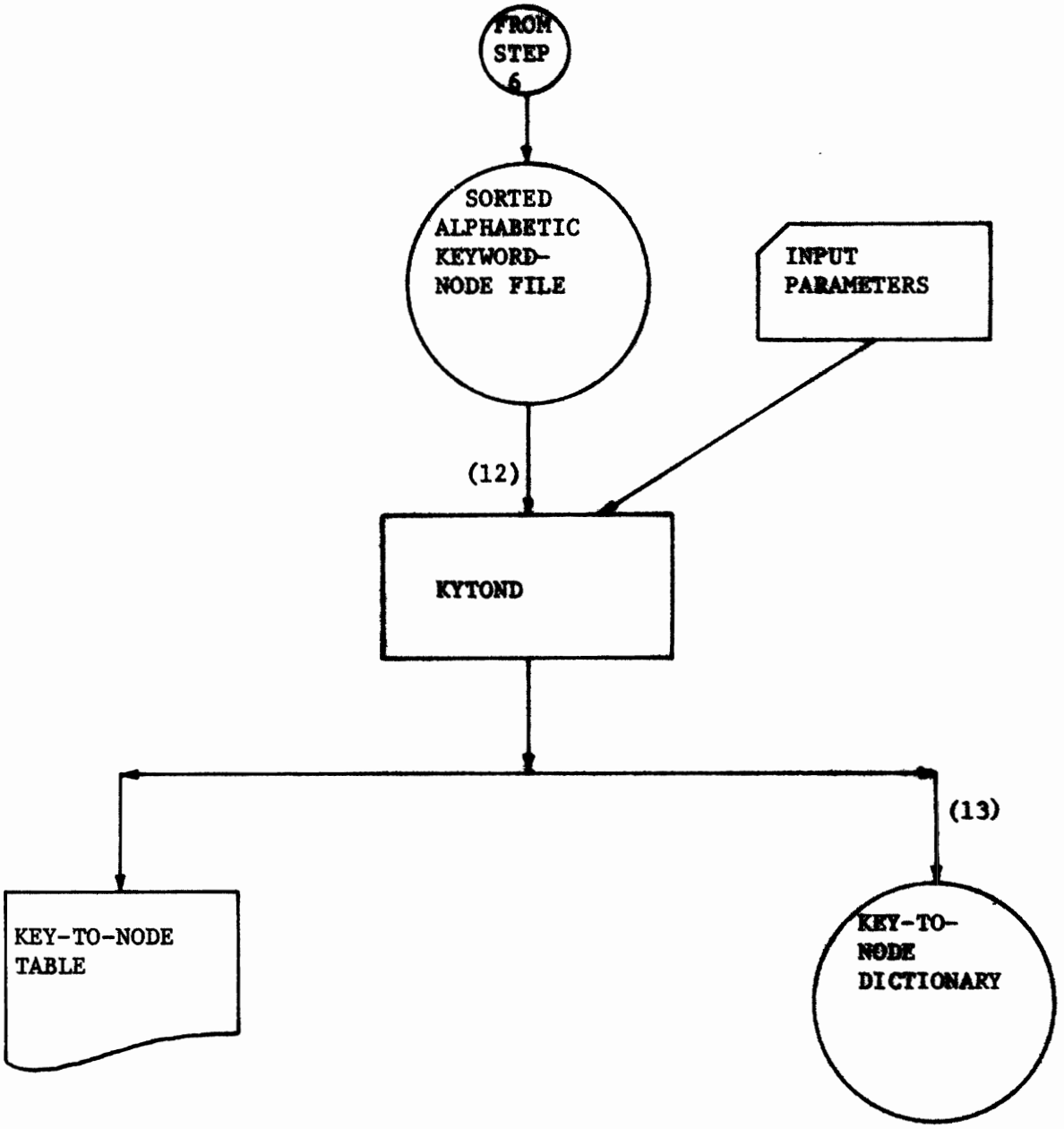
PROGRAM NAME: KYTOND

OBJECTIVE: To create the key-to-node dictionary and table.

The key-to-node dictionary was created for the experimental data base. A sample of the key-to-node table is shown in Figure 3.19. The number of unique keys in the key-to-node table is 5053. These 5053 keys are equivalent to those keys created in the indexing process.

KEY	NODE
ABYSS	1•2•2•1•1*1•2•2•2•1*1•2•3•2•1*
ACADEMY	1•2•1•1*1•3•1•2*1•3•2•1*1•1•2•3•3*1•1•3•1•2*1•1•3•1•3*1•1•3•2•2*1•2•2•1•1*1•2•2•1•3*1•2•3•2•3* 1•2•3•3•2*1•2•3•3•3*1•3•2•2•1*1•3•2•3•2*1•3•2•3•3*
ACAI	1•2•3•2•1*
ACCELERATE	1•3•1*1•1•3•3*1•2•1•1*1•3•3•1*1•1•1•1•1*1•1•1•3•2*1•1•3•1•3*1•2•1•2•1*1•2•1•3•1*1•2•1•3•3* 1•2•3•2•3*1•3•2•3•2*1•3•3•2•3*
ACCIDENT	1•2•1•1*1•3•1•1*1•3•1•2*1•1•2•2•1*1•1•3•3•3*1•2•3•1•2*1•2•3•2•1*1•3•2•2•1*
ACCOMMODATE	1•1•1•3•1*1•3•3•2•3*
ACCOMPLICE	1•3•1•2*1•3•1•3*1•3•2•1*1•1•1•3•2*1•1•2•2•1*
ACCRA	1•1•1•2•1*1•1•3•3•2*
ACCREDIT	1•1•1•3•2*1•1•3•3•3*1•3•2•3•2*
ACCRU	1•2•1•1*1•3•1•1*
ACCUMULATE	1•3•1•1*1•3•1•3*1•1•2•2•1*1•1•3•1•1*1•1•3•3•3*1•2•2•3•2*1•2•3•3•1*1•3•2•3•2*1•3•3•2•1*1•3•3•3•1*
ACCURATE	1•2•1•1*1•3•1•2*1•3•1•3*1•2•2•1•2*1•2•2•3•1*1•2•2•3•2*1•2•3•1•2*1•2•3•2•1*1•3•2•2•2*
ACCUSE	1•2•1•1*1•3•1•1*1•3•1•2*1•1•1•1•1*1•1•1•1•2*1•1•1•3•2*1•1•1•3•3*1•1•2•1•1*1•1•3•2•1*1•2•2•1•1* 1•2•2•3•3*1•2•3•2•3*
ACCUSTOME	1•2•2•2•2*1•2•3•3•2*
ACCUS	1•2•1•1*1•2•1•2•3*1•2•2•3•2*
ACOSTA	1•2•1•1*1•2•3•3•2*
ACQUAINT	1•3•1•2*1•1•2•2•3*1•1•2•3•1*1•1•3•2•1*1•1•3•2•3*1•3•3•3•3*
ACQUISIT	1•1•3•1•1*1•3•3•2•1*
ACTION	1•2•3•2•3*
ACTIVATE	1•3•3•1*1•3•3•2•1*
ACTIVE	1•3•1*1•1•1•3*1•1•2•2*1•1•3•1*1•2•1•1*1•2•1•2*1•3•2•1*1•3•3•1*1•3•3•2*1•1•1•1•2*1•1•1•1•3* 1•1•1•2•1*1•1•2•1•3*1•1•2•3•1*1•1•3•2•2*1•1•3•3•3*1•2•1•3•1*1•2•1•3•2*1•2•2•1•1*1•2•2•1•2* 1•2•2•2•2*1•2•2•3•2*1•2•3•1•2*1•2•3•2•1*1•2•3•2•3*1•3•2•3•2*1•3•2•3•3*1•3•3•3•1*
ACTIVITY	1•3•1*1•1•1•2*1•1•3•1*1•2•1•1*1•2•1•3*1•2•2•2*1•2•2•3*1•3•2•1*1•3•2•2*1•3•3•1*1•3•3•2*1•1•1•1•1* 1•1•1•1•3*1•1•1•3•2*1•1•2•1•2*1•1•2•2•2*1•1•2•2•3*1•1•2•3•3*1•1•3•2•2*1•1•3•2•3*1•1•3•3•1* 1•1•3•3•3*1•2•1•2•2*1•2•1•2•3*1•2•2•2•1•1*1•2•2•1•2*1•2•3•1•1*1•2•3•1•2*1•2•3•2•2*1•2•3•2•3*

Fig. 3.19



MACRO FLOW CHART FOR STEP 7
PROGRAM NAME: KYTOND

STEP 7 Quick Reference Chart

Program Name: KYTOND

Function	Section	Name (if any)	Reference in Manual
Input	3.8.2	Sorted Alphabetic Keyword-Node File (SKYNDE)	Table 3.11 pg. 195
Output	3.8.3	Key-To-Node Dictionary (KEYNDE)	Table 3.14 pg. 204
Input Parameters	3.8.4		Table 3.15 pg. 206
Output Report	3.8.5	Key-To-Node Table	Fig. 3.14 pg. 208

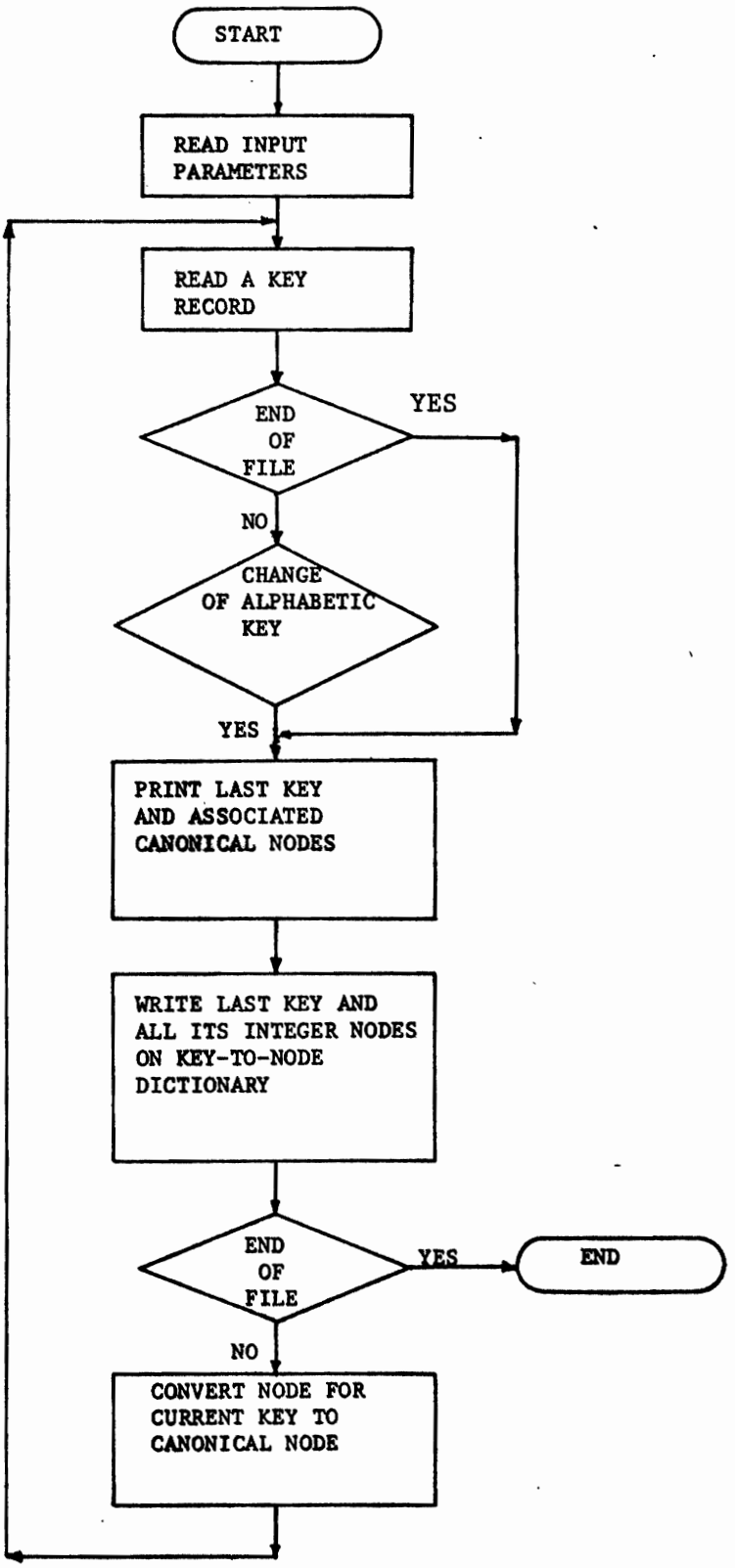
*Names in parentheses indicate names used in Job Control Language.

```
/LOGON LANG,D1035,TIME=8000,PRIORITY=7

/ERASE SKYNDE

/STEP

/FILE SKYNDE,LINK=DSET12,RECFORM=F,RECSIZE=46,BLKSIZE=4094,                -
/FCBTYPE=BTAM,DEVICE=T9N,STATE=FOREIGN,VOLUME=SKYNDE,OPEN=INPUT

/ERASE KEYNDE

/STEP

/FILE KEYNDE,LINK=DSET13,RECFORM=V,BLKSIZE=4096,FCBTYPE=BTAM,              -
/DEVICE=T9N,VOLUME=KEYNDE,OPEN=OUTPUT

/EXEC LMKYTOND

*00010005003020

/LOGOFF
```

*Input Parameter Card

Job Control Language to Execute Step 7

Program Name: KYTOND



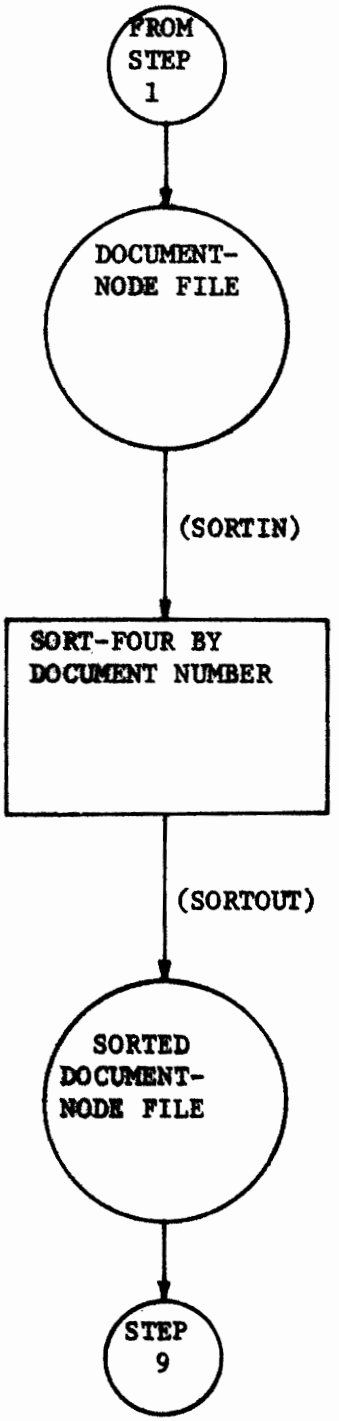
DETAIL FLOWCHART FOR KYTOND

3.11.8 STEP 8 SORT FOUR

PROGRAM NAME: SORT

OBJECTIVE: To sort the document-node file in document number order

The 1669 records on the document-node were sorted in document number order by the SPECTRA system sort. This file could now be used to create the final classified document file for the experimental data base.



MACRO FLOW CHART FOR STEP 8
PROGRAM NAME: SORT-FOUR

STEP 8 Quick Reference Chart

Program Name: SORT - FOUR

Function	Section	Name (if any)	Reference in Manual
Input	3.9	Document - Node File (DOCNDE)	Table 3.6 pg. 164
Output	3.9	Sorted Document Node File (MRGIN)	Table 3.6 pg. 164

*Names in parentheses indicate names used in Job Control Language.

```
/ERASE DOCNDE

/STEP

/FILE DOCNDE,LINK=SORTIN,RECFORM=F,RECSIZE=20,BLKSIZE=4080,                -
/FCBTYPE=SAM,DEVICE=T9N,VOLUME=DOCNDE,STATE=FOREIGN,OPEN=INPUT

/ERASE MRGIN

/STEP

/FILE MRGIN,LINK=SORTOUT,RECFORM=F,RECSIZE=20,BLKSIZE=4080,                -
/FCBTYPE=SAM,DEVICE=T9N,VOLUME=MRGIN,OPEN=OUTPUT

/EXEC SORT

SORT FIELDS=(5,4,A),FORMAT=FI

END

/LOGOFF
```

Job Control Language to Execute Step 8

Program Name: Sort-Four

3.11.9 STEP 9 CREATE FINAL CLASSIFIED FILE

PROGRAM NAME: MRGCLY

OBJECTIVE: To create a final classified file, including document text, terminal node assignment and alphabetic keywords associated with the document.

A final classified file was created for the 1669 documents in the experimental data base. A sample of one document is shown in Figure 3.20.

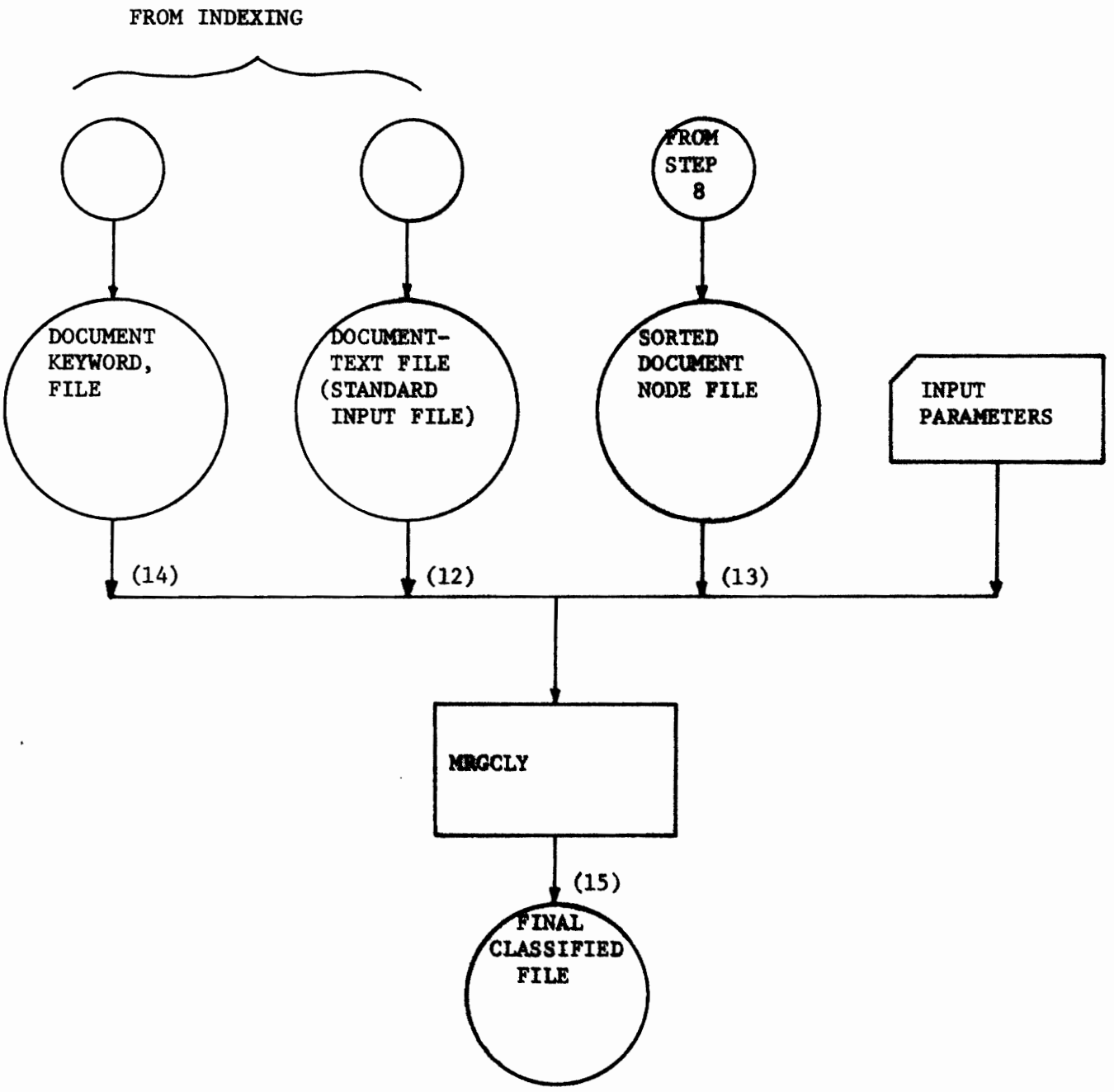
1.2.1.1 CELL NUMBER
0080 DOCUMENT NUMBER

((MSGNO 103 ??? *06/08/71* *09:15*
FBIS 22
ZEM_ UUK:DSY/DEG/FLY/FORP-D/NEWS.
TXXT_ PRAVDA'S DEMCHENKO ON ARAB FEDERATION
MOSCOW TASS INTERNATIONAL SERVICE IN ENGLISH 0709 GMT 25 APR 71 L
(TEXT) MOSCOW, APRIL 25, TASS--THE PROCLAMATION OF THE FEDERATION
OF ARAB REPUBLICS "IS A REFLECTION OF THE MOUNTING ANTI-IMPERIALIST
SENTIMENTS IN THE ARAB EAST AND THE DEEPGOING SOCIAL PROCESSES UNDERWAY
SENTIMENTS IN THE ARAB EAST AND THE DEEPGOING SOCIAL PROCESSES UNDERWAY
IN THE COUNTRIES OF THAT AREA", WRITES IN "PRAVDA" ITS COMMENTATOR
PAVEL DEMCHENKO. HE NOTES THAT THE POINT AT ISSUE IS IN THE
FIRST PLACE THE CONSOLIDATION OF FORCES SUPPORTING THE COURSE FOR THE
CONSTRUCTION OF SOCIALISM IN PROSPECT. THE DECLARATION ON THE
ESTABLISHMENT OF THE FEDERATION DIRECTLY SAYS THAT IT WILL BE THE
KERNEL OF A UNITED ARAB SOCIALIST SOCIETY.
PARTICIPANTS IN THE FEDERATION DECLARED THAT THEY WOULD PURSUE A
COORDINATED FOREIGN POLICY AND THAT THEIR PRIME TASK IS THE LIBERATION
OF ALL THE OCCUPIED ARAB LANDS. "THUS," DEMCHENKO STRESSES, "THE
CONSISTENT AND CONSTRUCTIVE LINE OF THE UNITED ARAB REPUBLIC AIMED
AT A POLITICAL SETTLEMENT OF THE MIDDLE EAST CRISIS RECEIVES A NEW
IMPETUS AND WILL RELY FOR SUPPORT ON A MORE SOLID BASE IN THE ARAB
WORLD ITSELF. PROGRESSIVE REGIMES IN ARAB COUNTRIES HAVE
STRENGTHENED".
NO WONDER, THE AUTHOR WRITES, THAT THE CONSOLIDATION OF THE
ANTI-IMPERIALIST FRONT OF ARAB STATES WAS MET WITH AN ALARM IN
WASHINGTON AND TEL AVIV. THE RIGHT-WING ISRAELI AND U.S.
PRESS KICKED UP A ROW ABOUT WHAT THEY DESCRIBE AS A "TOUGHENING"
OF THE ARADK AND. "ALL THIS," DEMCHENKO NOTES, "REFLECTS THE
ASPIRATION OF ISRAEL'S RULING CIRCLES BACKED BY THE USA TO USE ANY
PRETEXT FOR PROTRACTING THE CONFLICT".
"NOW" THE COMMENTARY SAYS, "WHEN THE STRENGTHENING UNITY BETWEEN
THE UNITED ARAB REPUBLIC, SYRIA AND LIBYA HAS ACQUIRED A CONCRETE
FORM, BETTER PROSPECTS FOR THE USE OF THEIR POSSIBILITIES
IN THE INTERESTS OF PEACE, CREATIVE ENDEAVOUR AND PROGRESS IN ALL
SPHERES OF LIFE HAVE OPENED UP BEFORE THE PEOPLES OF ARAB
COUNTRIES".
25 APR 0925Z GFE
NNNN
NNDD

KEYS ASSIGNED TO DOCUMENT NUMBER 0080

ANTIIMPERIALIST	APRIL	ARAB	AUTHOR	BASE	CIRCLE
CONCRETE	CONFLICT	CONSOLIDATE	COUNTRY	COURSE	CREATE
CRISIS	DEMCHENKO	EAST	ENGLISH	ESTABLISH	FEDERATE
FORCE	FOREIGN	FORM	FRONT	INTERNATIONAL	ISRAEL
ISSUE	KERNEL	LAND	LIBERATE	LIBYA	MIDDLE
MOSCOW	OCCUPY	PARTICIPATE	PAVEL	PEACE	POLICY
POLITICAL	PRAVDA	PRESS	PROGRESS	REGIME	REPUBLIC
RULE	SETTLEMENT	SOCIALIST	SOCIAL	SOCIETY	SPHERE
STATE	STRENGTH	SUPPORT	SYRIA	TASK	TASS
THU	UNITE	USA	US	WASHINGTON	

Fig. 3.20



MACRO FLOW CHART FOR STEP 9
PROGRAM NAME: MRGCLY

STEP 9 Quick Reference Chart

Program Name: MRGCLY

Function	Section	Name (if any)	Reference in Manual
Input	3.10.2	1. Document Text File (TEXTF)	Fig. 20 pg. 14
		2. Sorted Document Node File (MRGIN)	Table 3.6 pg. 164
		3. **Unique-Word-Within Document** (UNQKEY)	Fig. 2.10 pg. 36
Output Files	3.10.3	Final Classified File (CLFILE)	Table 3.16 pg. 211
Input	3.10.4		Table 3.17 pg. 214

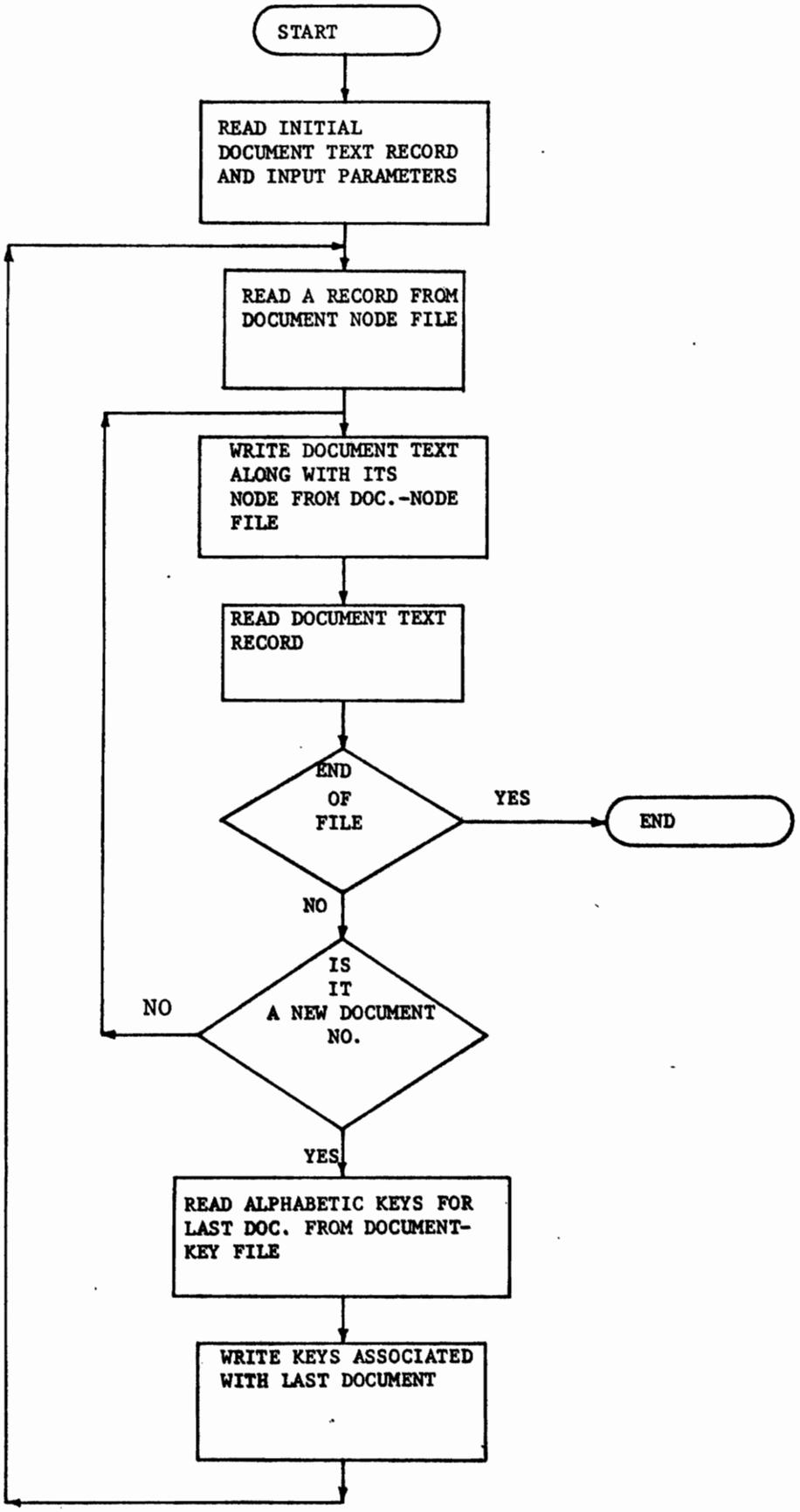
*Names in parentheses indicate names used in Job Control Language.

```
/LOGON LANG,D1035,TIME=20000,PRIORITY=7

/ERASE CLFILE

/STEP

/FILE CLFILE,LINK=DSET15,RECFORM=V,BLKSIZE=4096,FCBTYPE=BTAM,                -
/DEVICE=T9N,VOLUME=CLFILE,OPEN=OUTPUT

/ERASE UNQKEY

/STEP

/FILE UNQKEY,LINK=DSET14,RECFORM=F,RECSIZE=50,BLKSIZE=4050,                  -
/FCBTYPE=BTAM,DEVICE=T9N,VOLUME=UNQKEY,STATE=FOREIGN,OPEN=INPUT

/ERASE MRGIN

/STEP

/FILE MRGIN,LINK=DSET13,RECFORM=F,RECSIZE=20,BLKSIZE=4080,                   -
/FCBTYPE=BTAM,DEVICE=T9N,VOLUME=MRGIN,STATE=FOREIGN,OPEN=INPUT


/ERASE TEXTF

/STEP


/FILE TEXTF,LINK=DSET12,RECFORM=V,RECSIZE=4088,BLKSIZE=4096,                 -
/FCBTYPE=BTAM,DEVICE=T9N,VOLUME=TEXTF,LABEL=NSTD,                            -
/STATE=FOREIGN,OPEN=INPUT

/EXEC LMMRGE

*02000010

/LOGOFF
```

*Input Parameter Card

Job Control Language to Execute Step 9

Program Name: MRGE



DETAIL FLOWCHART FOR MRGCLY

CHAPTER 4

SUGGESTIONS FOR THE USER

The semi-automatic indexing and automatic classification procedures described in this report can be used on any data base, on any machine, but before the procedures are begun, the user should first consider the information in the following sections.

4.1 Input File--Text, Abstracts, or Title Words?

The semi-automatic indexing procedure can generate index terms for any file, whether it contains full text, abstracts, or title words. But in terms of time, both man and machine, it is worthwhile to consider for a moment the differences among the three types of files.

First, it must be pointed out here that this indexing method is semi-automatic. This means that human involvement and judgement at some point, enters into the automatic process of creating index terms. In this system of semi-automatic indexing a large number of terms are selected to describe a document. There is no automatic method of shortening the list of descriptors for one document. This human involvement takes time and in terms of this particular method of indexing the user should be aware of the extra human effort needed to index a complete document text. Abstracts or title words (containing fewer words to begin with), would require far less human intervention. (Further work is presently being done to obtain fewer but still pertinent index terms for each document - Section 4.3)

Remembering that the ultimate goal of indexing and classification

is to improve the effectiveness of retrieval, the user might at this point be wondering which type of file will do the most to improve retrieval. This is a somewhat unanswered question. It has not really been proven that full text indexing improves the results obtained sufficiently enough to override the cost incurred (be it man or machine) in the indexing of the full text. [9] It has been shown however, that abstract indexing does produce more efficient and more comprehensive retrieval terms than those created by just title word indexing. [9]

The user must decide for himself which type of file to use, but it is suggested that if the method in this report is used, abstracts or titles should be strongly considered, due to the reason mentioned above. If it is desired to use some other method of indexing, it is completely acceptable to enter this system with an already indexed file (see Section 2.1). The alphabetic index terms would only have to be converted to an integer representation (done by this system), and classification could begin immediately.

4.2 Choice of Machine

No matter what the advantages obtained through indexing and classification, the machine time used to obtain the end result must be reasonable, if these advantages are to be recognized. Both indexing and classification are lengthy processes which should only be done on a high speed computer such as the IBM 370/145. It is strongly suggested that the system be run in a dedicated environment. That is, an environment where only one job will be using the facilities of the computer. This means that your job will not be

slowed down due to the queuing of jobs waiting to use the resources of the computer. In the process of indexing and classifying the experimental data base described in this report, approximately 200,000 CPU seconds were used on the RCA Spectra 70/46. The amount of real time (length of time to run a job, calculated from the time it enters the machine until the time it leaves the machine) was not calculated. The Spectra 70 is a time sharing system allowing many jobs to run simultaneously and compete for the use of its resources. Consequantly, most of the indexing and classification of the experimental data base was not run in a dedicated environment. If it were, this large number of CPU seconds used might have been cut, and certainly the amount of real time used would have been cut considerably.

At this time, it is important to note that indexing and classification need be done only once, and it is the objective of this system to produce results which provide advantages over the long run that outweight the initially large startup cost.

4.3 Further System Enhancements

There are two particular areas in this system which might be worthwhile areas in which to consider further development.

One is that of restart and recovery procedures. As the system now stands, all restart and recovery procedures are machine dependent and are not automatically executed from within the indexing and classification system. Since many of the steps in this system are long, it would be worthwhile to install automatic restart procedures in some of the programs, in particular CLASFY.

A restart procedure could be developed which would allow the user to automatically stop and start CLASFY at any point in the classification. An automatic recovery procedure would enable the user to restart CLASFY where it ended in case of a machine failure. This procedure would protect against the possibility of lost time if the failure did occur. The addition of restart and recovery procedures would facilitate the running of this system.

A second area, and one which would take more time to develop, would be that of automatic indexing. An automatic indexing procedure could be developed which would eliminate the human involvement now necessary in the system. To do this, more sophisticated techniques such as syntactic analysis of sentences and dictionary construction (such as phrase dictionaries, synonym dictionaries, etc.), would have to be developed. Work is presently being done in text processing at the University of Pennsylvania in areas such as phrase recognition in order to make indexing more automatic. The goal would be to automatically process any document collection. That is, with the addition of complete automatic indexing to the system, any document collection could be processed and readied for use in a retrieval system without human involvement. This could be an important factor in the search for better methods of handling the information explosion noted in the introduction to this paper.

DISTRIBUTION LIST

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 copies
Office of Naval Research Department of the Navy Information Systems Program Code 437 Arlington, Virginia 22217	2 copies
Office of Naval Research Branch Office/Boston 495 Summer Street Boston, Massachusetts 02210	1 copy
Office of Naval Research Branch Office/Chicago 536 South Clark Street Chicago, Illinois 60605	1 copy
Office of Naval Research Branch Office/Pasadena 1030 East Green Street Pasadena, California 91101	1 copy
Director, Naval Research Laboratory ATTN: Library, Code 2029 (ONRL) Washington, D. C. 20390	6 copies

U.S. Naval Research Laboratory
Technical Information Division
Washington, D. C. 20390 — 6 copies

Commandant of the Marine Corps (Code AX)
Dr. A. L. Slafkosky
Scientific Advisor
Washington, D. C. 20380 — 1 copy

Office of Naval Research
Code 455
Arlington, Virginia 22217 — 1 copy

Office of Naval Research
Code 458
Arlington, Virginia 22217 — 1 copy

Naval Electronics Laboratory Center
Computer Science Department
San Diego, California 92152 — 1 copy

Naval Ship Research & Development Ctr.
Dr. G. H. Gleissner
Computation & Mathematics Department
Bethesda, Maryland 20034 — 1 copy

Office of Naval Research
New York Area Office
Dr. J. Laderman
207 West 24th Street
New York, New York 10011 — 1 copy