United States
Naval Postgraduate School

THESIS

APPLICATION OF A DATA STRUCTURING CONCEPT
IN A GENERAL-PURPOSE FACT-RETRIEVAL SYSTEM

by

Richard Joseph Petrucci

September 1970

92

Application of a Data Structuring Concept in a General-Purpose Fact-Retrieval System

by

Richard Joseph Petrucci
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1961

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1970

Author Richard J. Petrucci

Approved by: Gary Arlen Kildall
Thesis Advisor

[illegible]
Chairman, Committee for Computer Science

Milton U. Clauser
Academic Dean

ABSTRACT

An on-line, general-purpose, fact-retrieval system is presented which employs a classificatory data structuring technique. The technique embraces the basic concept of hierarchical classification of data and provides users with multiple avenues of access to a data file. Additionally, the data file may be partitioned into unrelated data sets.

TABLE OF CONTENTS

page blank

LIST OF FIGURES

page blank

I. INTRODUCTION

The term "information retrieval" and the initials "IR" were coined by the editors of Fortune about ten years ago. However, Vannevar Bush first formally declared the necessity for an information retrieval discipline in his "As We May Think" article which was written for Atlantic Monthly in 1946. The United States Government and those people involved in Library Science were truly the first innovators of this discipline in the mid-fifties. The technological explosion being felt at that time prompted government agencies and library scientists to search for more efficient systems for indexing, storing, and retrieving documents. Primary concern was the assurance that vital technical information would be available to all possible users. The discipline of information retrieval as we know it today emerged as a result of this technological explosion.

Information retrieval has been defined in numerous ways. However, all definitions share a common point which is best stated by Taube [Ref. 1] as: "The right information made available to the right person at the right time." Bourne [Ref. 2] states that "Information retrieval has become a generic term, firmly established through common usage, which includes reference, fact, and document retrieval." Bourne also differentiates between data processing and information retrieval. The former includes the manipulation, replacement, alteration, or addition to the data on file while the later is concerned with the storage of data in unaltered form for later re-use. Use of the term "information retrieval" in this paper implies the generic meaning stated by Bourne.

This paper is devoted to the investigation of a data structuring concept proposed by Kildall [Ref. 3] for use in a general-purpose fact-retrieval system. Before investigating Kildall's proposal in section VI, the techniques of indexing, storage, and retrieval established for Library Science purposes will be reviewed. These basic techniques form a foundation for the design of specific IR systems.

Information retrieval is divided into three major operatives:

1. Indexing (classification, description, and structuring of information sources).

2. Storage (organization and storage of files).

3. Retrieval (searching and displaying information).

Figure 1 is a simplified diagram which illustrates a typical information retrieval process. An index is constructed which describes the information source (document or record) and is stored in a file along with the source itself. A request for information (query) is directed to the index file where the location of the requested document within the information file is found. A search of the information file then results in the retrieval of the document. This process is analogous to the indexing and storing of new books received in a library, and the search for information by a library patron.

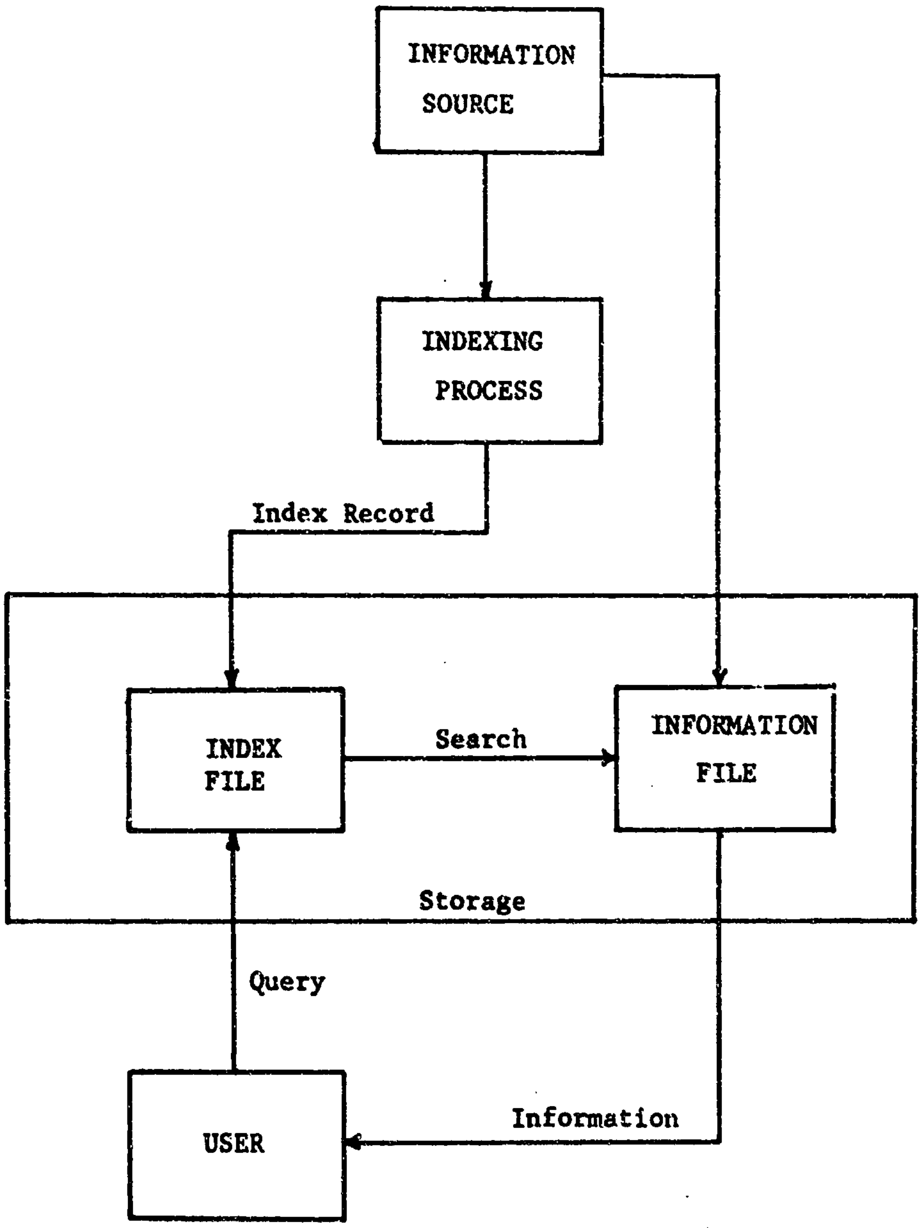


Figure 1

Basic Flow Diagram of the Information Retrieval Process

II. INDEXING

Indexing is the classification, description, and structuring of information in such a manner that retrieval of the information is accomplished expeditiously. This task is performed on information sources such as books, documents, and files and is an integral part of the information retrieval process. Since retrieval is the counterpart of indexing, the indexing and retrieval schemes used in an IR system must be compatible in order for a user to communicate with the system. Clearly, retrieval efficiency (i.e., ease and speed of retrieving desired information with a minimum of false drops[1]) is related to the efficiency and consistency of the indexing process.

As a rule, the information base of an IR system is specialized and as such requires a professional jargon. Ideally, the indexer and system user are experts in this professional language. However, this may not necessarily be true and causes a problem commonly confronted by IR system designers. The problem is how to structure specialized data for input to the system in a manner that is convenient to both the indexer and user while maintaining data accessibility. An example of an indexing language is the Dewey Decimal System used for indexing library books.

Selection of an indexing language is based upon the following considerations:

[1]Output of irrelevant information as a result of a retrieval request is called a "false drop."

1. The language should be convenient to use, such as natural language or a language that could be easily learned.

2. Computerized systems require that the language be rigid enough to be usable in the machine but must also remain convenient for human utility.

3. The vocabulary should be broad enough to allow accurate description of the information.

4. The language should be flexible enough to allow modification as changes in information occur.

There are numerous indexing languages in use today each tailored to suit specific usage of the IR system. Therefore, indexing languages normally reflect the viewpoint of the system designer in his attempt to organize the system's data base to best suit the needs of the user. Several indexing techniques which evolved from Library Science will be reviewed in the sections that follow. These techniques appear to form the nucleus from which specialized systems are formed. Although the techniques are primarily oriented toward document indexing, variations are used in all types of IR systems. The techniques are presented in ascending order of:

1. Effort on the part of the indexer.
2. Difficulty in automating.
3. Indexing power
4. Retrieval efficiency.

A. UNIT-TERM INDEXING

The simplest indexing technique involves the extraction of descriptive words from the information source. The source is then associated with each of the terms used to describe its content. In the case of a library book, or other document, descriptive words may be taken from the title, abstract, or the text itself. This technique requires a minimum of effort (other than reading the source) on the part of the indexer. In addition, the indexing is accomplished rather quickly since the indexer need not be ultimately familiar with the subject material. Unit-term indexing is particularly advantageous when no information is available on the spread of subject material. The addition of new material to the data base is easily accomplished by expanding the vocabulary (unit-terms) to include new descriptive words. However, unit-term indexing lacks rules for combining terms into units which have meaning. This shortcoming causes indexing problems when synonyms, plural word forms, and generically related terms are encountered in the source document.

The search device used in such a system is an alphabetical listing (indexing record) of the key words used by the indexer. In general, the information source is listed with each key word and is used as a source descriptor, or the listing may indicate the location of the source, or both. It is possible that the user will have difficulty in using this system unless he knows precisely the topic that he is searching for. An analogy may be drawn to searching the telephone book for a name when the spelling of the name is not known. Therefore, this indexing scheme is often utilized in IR systems where the user is familiar with the professional jargon contained in the information sources (e.g., technical libraries).

An excellent example of this subject-indexing[2] technique is the Uniterm Coordinate Indexing System which dates back to 1952. The Uniterm ("unit-term") System includes fifteen rules governing the indexer's operation, rules for determining key words, methods for processing word meanings, and cross-referencing techniques. Some agencies using this system have drafted standard unit-terms (key words) to be used by indexers. However, this is unnecessary for an unstructured language since new unit-terms may be added without perturbing the existing system. An example of an index that might be constructed from a Uniterm System is shown below. The numbers below the unit-terms might represent reference serial numbers, or library call numbers.

ABLATION

452 573 772

ADSORPTION

137 459 823 1201

ADHESIVE

491

AERODYNAMIC

139 241 242 357 552 1010 1168

---

[2]"Subject indexing," "keyword indexing," and "coordinate indexing" are terms commonly used to describe the technique presented here.

B. KEY-WORD-IN-CONTEXT INDEXING

Another very common subject indexing technique is called "Key-Word-In-Context" (KWIC) indexing[3]. The indexing power of KWIC is very slightly greater than the simplest of subject indexing techniques since the key word is shown in the context of the entire subject. There are several variations in KWIC format but essentially it is an alphabetical listing of key words. Whole phrases are extracted from the source so that a user can easily determine the role of the key word. The distinguishing feature of KWIC is its display format shown in the example below. Let us suppose that the title of a source document is: "Principles of Automated Information Retrieval." Assuming that the indexer selects four key words to describe the source, the KWIC index would appear as:

"5135 Principles of <u>AUTOMATED</u> Information Retrieval
iples of Automated <u>INFORMATION</u> Retrieval 5135 Princ
ion Retrieval 5135 <u>PRINCIPLES</u> of Automated Informat
omated Information <u>RETRIEVAL</u> 5135 Principles of Aut"

Note that "automated", "information", "principles",and "retrieval" are individual key words. A user desiring this source document could find it by using any one of the four key words. Note also that a user may find this system easier to use than the Uniterm System if he is unfamiliar with the subject material.

[3]Also referred to as "permuted" or "permuted title" indexing.

C. THESAURUS

Indexing power may be increased further by determining generic relationships between key words. The Armed Forces Information Agency (ASTIA) and the Defense Documentation Center (DDC) have produced thesauri which are alphabetical lists of indexing terms with related terms and "see" references. These lists are used by indexers as means of standardizing their operation. In other words, indexers describe similar information sources in consistent fashion. These thesauri define some hierarchy in key words and are useful to the user as well as indexer since they allow the user to formulate queries with the exact terms used by the indexer. An example of a thesaurus borrowed from Meadow [Ref. 4] is exhibited below.

COMPUTERS

(Computers and Data Systems)

Includes:

Calculating machines

Generic to:

ANALOG COMPUTERS

ANALOG-DIGITAL COMPUTERS

BOMBING COMPUTERS

.

.

.

Also see:

DATA PROCESSING SYSTEMS

.

.

.

SIMULATION

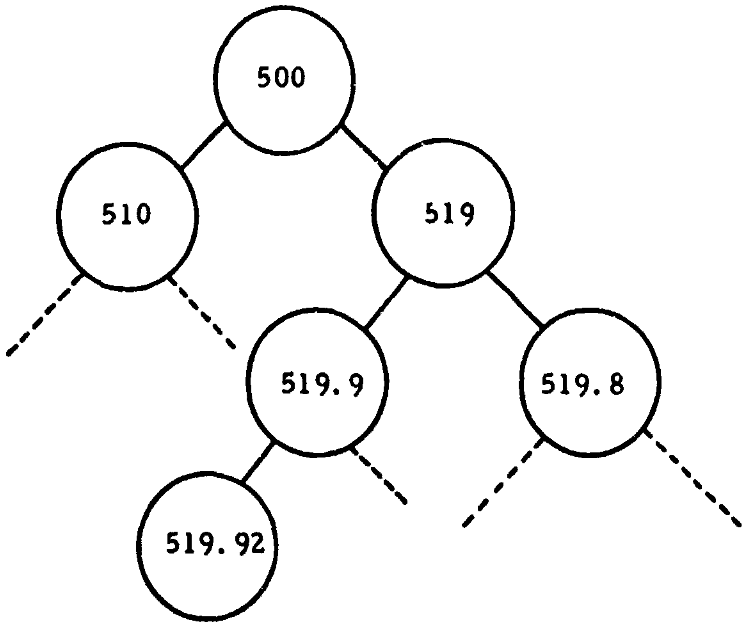
Computing gun sights use GUN SIGHTS

D. HIERARCHICAL CLASSIFICATION

Probably the most widely used indexing technique is that of hierarchical classification where a universe of information is repeatedly divided and sub-divided into a classificatory tree. This index language has a very tightly controlled but simple vocabulary contained in an authority list of key words provided with the classification system. Each key word in the authority list is assigned a numeric or alphanumeric code (mnemonic codes could be used but normally are not). As can be seen in the tree structure exhibited below, a key word contains all those key words generic to it (i. e., above it in the branch of the tree from which it was derived). Hierarchical schemes allow the indexer to describe an information source in generic levels so that the user may formulate his query in more general or more specific terms by moving up or down the classification tree.

Modification of key word meaning is difficult to accomplish since changing one word in the tree affects all key words generic to it. However, changes at the bottom of the tree are easily made since no perturbation of the tree occurs. Expansion of the vocabulary used in this sytem is readily accomplished by expanding the tree horizontally.

The most well known hierarchical systems are the Dewey Decimal Classification System (exhibited below), the Library of Congress System, and the Universal Decimal Classification System.



500	Pure Science
510	Mathematics
519	Probabilities and Statistical Mathematics
519.9	Treatment of Data
519.92	Programming (linear and dynamic)

E. FACETED INDEXING

In the immediately preceding section a classification technique was presented which structures a topic (universe of information) by dividing and subdividing it to form a classificatory tree. Faceted indexing deals with individual key words taken from the data source and grouped into categories with respect to their usage within the source. Terms within each group are structured into a classificatory tree. A term extracted from the source is analyzed from several points of view and a group of indexing terms are synthesized to describe the key word in context. This technique is referred to as "facet analysis," "faceted indexing," and "relational indexing" where each key word's point-of-view-analysis is called a facet.

An excellent example of faceted indexing is given by Meadow [Ref. 4]. Let us suppose that "steel" is a key word taken from a source document. The document contains information relating to the manufacture, use, chemical analysis, and properties of steel. By appending descriptors to the key word "steel" the following index terms are created:

STEEL, manufacture of

STEEL, use in automobiles

•

•

•

These index terms are not predefined in any authority list but are constructed by the indexer by appending descriptors to the key word. The terms follow some syntactic rule such as: subject followed by modifier, followed by operation modifier. The utility of this technique is that the indexer, armed with a descriptor list and syntactic rules tailored to suit the particular IR system, may analyze a source from many points of view and construct index terms that describe the information content in great detail.

F. AUTOMATIC INDEXING

In the foregoing discussions, it was assumed that the indexer was human. A treatment of automatic (computer) indexing is now in order.

Automatic indexing is difficult to accomplish for two main reasons. First, the information source must be in machine readable form. In the case of books or other lengthy documents this is a very expensive requirement. However, development of character recognition devices and the production of transcripts in machine code as a by-product of automatic typesetting have eased the cost of this requirement. The second problem, and the more serious, is the development of algorithms or heuristics which derive meaning from string of characters. This is an area of Artificial Intelligence in which a good deal of research has been expended. However, the results of this research have been empirical since we lack sophisticated linguistic and semantic knowledge. References 5, 6, 7, and 8 contain excellent treatments of the research conducted and problems involved in machine translation of natural language while ref. 9 contains a comparison of manual and automatic indexing techniques.

There is an automatic indexing technique in commercial use today; however, it is a "brute force" adaptation of KWIC. Basically, the technique produces index key words by comparing words from the source to words stored in an authority list. There are many limitations to this system such as correct handling of hyphenated words, plural forms, and proper nouns but the primary limitation is that the list must contain a sufficient number of appropriate words in order for a source to be adequately indexed. The size, speed, and complexity of such a system should be obvious.

Referring to figure 1 it is seen that the indexing process produces index records. The contents of the records vary widely and are dependent upon the type of IR system (e.g., document, fact, or reference). In addition to subject descriptors, the index may contain the location of the information, source, author, reference to another index record, or other information deemed pertinent by the system designer. It will also be noted from the figure that the information source, or information concerning the source, will also be stored in the IR system. In the case of a large document such as a book, it probably will not be stored in the computer but rather a reference or abstract will be stored as a substitute. In some cases, the index record itself will contain all of the information associated with an information source. For example, an index record for a library book may contain the book's location within the library, therefore, the system will present the index record itself in answer to a user's que[illegible].

III. STORAGE

This section of the paper contains descriptions of various techniques used for organizing index and information files within an IR System's storage media. There will be no discussion of storage devices since it is assumed that the reader is already familiar with computer equipment. The reader is aware, of course, that the system's capacity, cost, and response time are greatly affected by the selection of various storage media.

A. FILE ORGANIZATION

Organization of an index file or information file specifies the positioning of the records in relation to one another within the file along with the physical position of the file within the storage media. Choice of a rule which governs file organization is dependent upon desired response time, peak retrieval loads, system reliability[5], category of users, cost, rate of information change, rate of system growth, and type of storage media. There are several rules for file organization which are extensively used in IR systems and they are presented here. These rules are equally applicable to index and information files.

1. Sequential Organization

The first method involves the sequential placement of records within a file. The $(i+1)^{st}$ record follows (physically and/or

[5] Ability to retrieve a maximum of information with a minimum of false drops.

logically) the i^{th} record. For example, the alphabetical listing of subject-indexing key words, alphabetical arrangement of employee records, etc. This method is very conservative of memory space since there is no need to supply pointers or links to indicate where the next record in the file is located. On the other hand, additions or deletions to the file are difficult to make. Let us suppose that we desire to add a new name to the telephone book. Then all of the names which follow the inserted name must be moved. Likewise, the deletion of a name results in perturbation of the list. This type of organization is most commonly used with magnetic tape where records are searched sequentially.

2. Chaining

Another technique of file organization is called "chaining" where addresses (links, chains, or pointers) are stored in one or more fields of a record to indicate the location of the next record within the file. Recall from the discussion of indexing that thesauri contain "see" references. These references are links which convey the idea of chaining. Chaining is a particularly effective method when used in a crowded memory since "referred to" records may be placed in any available space within the memory (unlike the rigid sequential scheme). Also, the utility of chaining is fully realized in a system which experiences a high rate of information change. This method requires more memory space than the sequential scheme since extra fields must be appended to the records to accommodate the links.

a. Branching

An extension of the chaining technique is referred to as a "branching structure." Branching is used to achieve versatility in changing record entries, changing file structures, and conversion, where possible, of variable-length records to fixed-length records. A trivial example is shown in Figure 2. which exhibits the idea of branching file structures.

Let us suppose that our file consists of all military flying clubs in the United States. Each record consists of the club's name, address (airport, city, state), membership, and type of aircraft. Obviously, these records are variable-length because the number of aircraft owned by each club is variable. The main file may be converted to fixed-length records by replacing the aircraft type fields with a single address. The aircraft types could then be included in another fixed-length file. The address in the main record links to an address file which in turn points to the file containing the aircraft types. Repetition of aircraft type is eliminated from the main records, main records are fixed-length, and changes are made only to the address file not the main file or aircraft file.

Figure 3 exhibits another feature of this technique which replaces all field entries in the main file (except the name) with addresses. If it is later decided to add "county" to "city" and "state" then no changes are required in the main file but a field must be added to each of the "city-state" file records to absorb the new addition.

MAIN FILE

⋮

	NAME	AIRPORT	ADDRESS	AIRCRAFT	NEXT RECORD
10	NALF MONTEREY	MONTEREY	MONTEREY CA.	100	32
⋮					
17	MINOR AFB	ABC INTERN'TL	NEEDLES CA.	101	28

⋮

ADDRESS FILE

100	210	211	213	214
101	213	214	215	

AIRCRAFT FILE

210	CESSNA 150
211	CESSNA 172
212	CESSNA 180
213	CHEROKEE 180
214	T-34
215	PT-19

Figure 2

Conversion of Variable-Length Records to Fixed-Length Records using the Branching Technique.

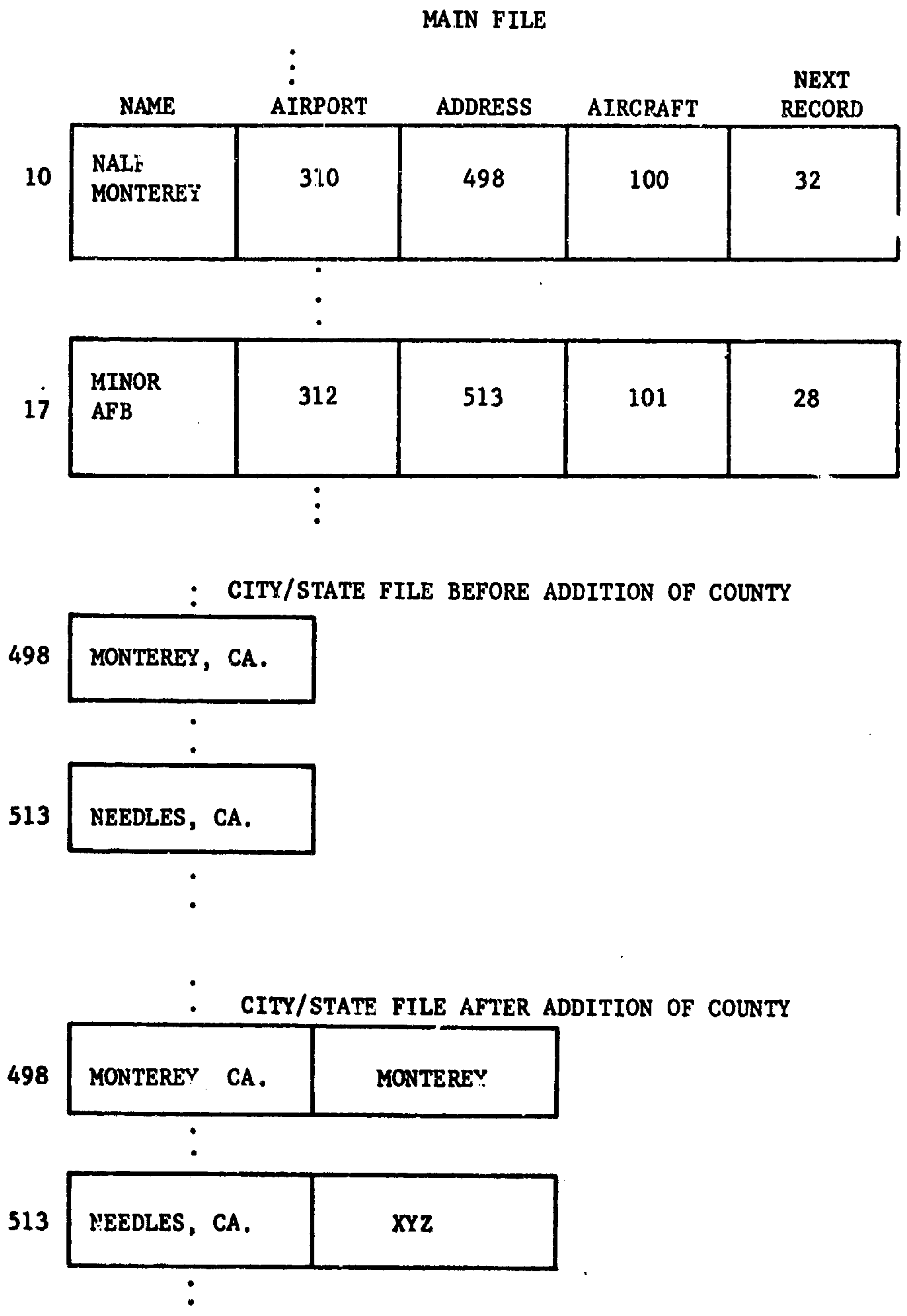


Figure 3

Addition of Records to an Existing Branching Structure

3. List Structuring

Although chaining and branching allow records to be scattered throughout memory, their membership in a particular file is maintained by some order of relative placement (e.g., employee records logically linked in alphabetical order but physically scattered throughout the file). List structuring does not require that records be ordered in any specific manner within a file. Further, the fields of a record may be physically separated and then linked to form a logical record. The advantage of this form of storage is the freedom of changing field content structure, record content, and file structure. However, this method requires a great deal more memory space than any other technique. In addition, the retrieval process is relatively slow since more time is required to gather the elements of a record together.

The three techniques of file organization described above are all forms of list structuring and each demonstrates a different degree of structural freedom. Chaining requires that fields remain contiguous, but records, while remaining ordered, may be physically separated. Branching is an extension of chaining allowing fields to contain address linkages to other fields. The last method allows any ordering and structuring of fields and records.

B. FILE SEQUENCING

It is important that records be sequenced (sorted) in some manner for use in IR systems. Sequencing is normally based on some particular attribute of a record (called a sort key) such as the "name" field of an employee record. Selection of the sort key is based on many considerations but the objective is to select the same sort key as may be used in a retrieval request. Subordinate sort keys may also be chosen when more than one record has the same primary sort key value (e.g., several employees with the same last name). Searching records which are ordered on the primary sort key is then called an "ordered search."

IV. RETRIEVAL

The retrieval process essentially consists of searching the index files and information files for information which satisfies a user's query. If the information is found, it is sent to the user, if not, the user is so informed. It should be noted that "searching" and "retrieval" are not synonymous. "Searching" is a file access operation used to locate records for matching against the query, while "retrieval" is the actual output of information which satisfies the query. However, use of the word "retrieval" here will imply the entire operation of searching and retrieval.

As previously discussed in section II, indexing and retrieval are counterparts since indexing refers to the structure of information for input to the files, while retrieval is the process of locating and displaying desired information. Therefore, the query language employed by the system user must be compatible with the index language employed by the system designer. It is important that the query and index languages use the same vocabulary in order for the IR system to understand the user's requests. The user must also be familiar with the system's logic in order to formulate an intelligent query. He must know if the system honors the use of Boolean relationships ("and," "or," "not") and magnitude comparators ("greater than," "less than," etc.) as query terms.

Once the query is formulated it is input to the system's index file. A matching process takes place at the index file where the terms used in the query are matched against the index file records. Index records which match the terms of the query are employed as locators to direct the retrieval of data from the information file.

The technique used in searching the index and information files is governed by the file organization (structure, sequencing, content, and storage medium). In the ensuing discussion of search techniques it should be borne in mind that whatever technique is used it is fixed within the IR system. Also, the interrelationship between search plan and file organization may limit file accessibility and search flexibility.

A. FULL-FILE SEARCH

One search plan incorporates a full-file search where every record of the file is matched (e.g., the value of the query term is matched against the value of the sort key). This plan is used when the order of records within a file is unknown (e.g., a file of employee records that are not alphabetically sorted). In this case, if we were searching for Doe's record and found Smith's it does not follow that we have searched too far since the records are not collated. In addition, there may not be any assurance that a single match satisfies the search (more than one Doe in the file). Therefore, all records within a file must be searched.

B. SEQUENTIAL SEARCH

A sequential search plan might be used when the records are not only sequenced but sequenced on the same term as is used in the query. Sequential searches are normally used in conjunction with sequential access type storage devices. The records of a file are matched sequentially until a successful match is made or when the value of the query term exceeds the value of the sort key. In this case, searching for Doe's record and locating Smith's record indicates that the search has not only gone too far but no successful retrieval will be made since there is no Doe in the file.

C. BINARY SEARCH

A binary search plan may also be used with a sequenced file. The term "binary" implies that a two valued decision is made after every match attempt. The search begins in the middle of the file. If the first match attempt is unsuccessful then the next attempt is made one-quarter file length away from the first. The direction of the subsequent search is dependent upon the result of comparing the value of the query term and the sort key (e.g., if the sort key is greater than the query term then move one-quarter file toward the beginning of the file). Each successive move is then made one-half the length of the preceding move. If there are n records in the file then there will be approximately $\log_2 n$ moves to exhaust the file.

D. DIRECT ACCESS SEARCHING

The last file searching technique relies upon a special type of index file called an inverted index. This is probably the most common type of index file used in IR systems. The inverted file records consist of the descriptors produced during the indexing process. The descriptors are used as sort keys for sequencing the records within the index. Appended to each descriptor field are fields which contain addresses of the associated records in the information file. Some type of search plan is conducted (usually binary) for matching descriptors (which are sort keys) to the query term. When a successful match is achieved, the addresses of the appropriate information records are obtained and the records are directly retrieved.

E. COMBINED SEARCH PLANS

The above treatment of search plans demonstrates that the techniques are dependent upon file organization but plans may be combined in one IR system. For example, a binary search may be employed in the index file to locate the disk and/or track which contains the desired information while a sequential search is made of the track for the requested records.

V. RETRIEVAL SYSTEMS

This section of the paper contains a discussion of the primary differences between reference, document, and fact retrieval in order to provide a frame of reference for the development of a fact-retrieval system. Reference retrieval is treated first since it is the least complicated of the three types of information retrieval.

Queries used in a reference-retrieval system contain only the topic for which information is desired (e.g., STEEL). The material provided to the requestor is a list of references pertaining to his topic.

Document retrieval queries are narrower in scope since descriptive terms are used to modify the topic (e.g., STEEL, chemical properties of). Documents are provided to the requestor which contain the desired information.

Fact-retrieval systems are the most complicated and powerful of all since they are capable of providing specific answers to specific questions.

A. REFERENCE RETRIEVAL

Reference retrieval is the first step taken by one in search of specific information. As explained above, a reference-retrieval system provides a user with a bibliography pertaining to the topic for which specific information is sought. The second step in the search for information is totally unrelated to the reference-retrieval system. The user must examine the documents listed in the bibligraphy in order to obtain the desired information. It is cle r that in the first step the user's search for information is narrowed from a search of the entire "library" to a "shelf" in the library.

B. DOCUMENT RETRIEVAL

The definition of document retrieval is not straight forward. One point-of-view holds document retrieval as the second step of reference retrieval. In another point-of-view, it is a special case of fact retrieval. What this author regards as document retrieval may be fact retrieval to another. The definition upheld by this author is the retrieval of unprocessed text word-for-word as it is stored in the information file. An example would be requesting a specific report from a technical library.

C. FACT RETRIEVAL

Fact retrieval ranges from the retrieval of processed text stored in an information file to the retrieval of specific answers to specific questions. The more powerful end of the spectrum is referred to as "question answering". Reference 10 contains an excellent treatment of the general characterizations, limitations, capabilities, and feasibility of the question-answering type of fact-retrieval systems. Reference 11 contains a practical example of a question-answering program.

Confusion arises at the low end of the fact-retrieval spectrum where it is difficult to distinguish the difference between document and fact retrieval. One point should help clarify the difference. Document-retrieval systems possess only rote memory which means that their capability is limited to the display of information word-for-word as it is stored in the data base. Fact-retrieval systems possess the capability of manipulating data stored in the data base into a form which best satisfies the user's request.

VI. DATA STRUCTURE FOR A FACT-RETRIEVAL SYSTEM

This section contains the description of a data structuring technique proposed by Kildall [Ref. 3] for use in a general-purpose fact-retrieval system. Specific useage of the system depends in part upon the type of information stored in its files. However, the nature of the system is the processing of data to provide a user with specific answers to his queries. Therefore, the system approaches "question answering." The data-structuring technique employs the basic concept of hierarchical classification which divides a topic (also referred to as a universe of discourse) into its class structure and correlates the data elements of the information file to a tree-type classificatory structure.

A treatment of the retrieval process is also provided here since the query format is directly related to the data-structuring technique.

This section is expressly devoted to a discussion of the data-structuring concept while section VII contains the description of the general-purpose fact-retrieval system which employs the proposed technique. The system was designed for the primary purpose of investigating the potential of the data-structure concept and not for production purposes.

As previously discussed, fact-retrieval systems range from the manipulation of processed text to "question answering." The system described herein maintains a position in the middle of this continuum. The term "general purpose" used here does not necessarily mean that the system may be utilized throughout the full range of fact retrieval. Rather, it means that the system will accommodate files which contain different types of information.

A. DATA STRUCTURE

The structure employed for indexing data incorporates the concept of hierarchical classification which allows the user to enter the data base in a number of ways in order to extract desired information. A universe of discourse is structured in terms of "classes" and a hierarchy of classes is established onto which the associated data elements are mapped. For example, assume that a universe of discourse consists of personnel records. The records consist of names, addresses, and telephone numbers which are members of the classes "NAME", "ADDRESS," and "TELEPHONE NUMBER." "NAME" is further divided into the subclasses "LAST," "FIRST," and "MIDDLE" while "ADDRESS" contains "STREET," "CITY," and "STATE." The data structure is then represented by a classificatory tree with the data elements related to the classes contained in the tree. The data element "DOE," for example, is identified as a member of the class "LAST," and the class "LAST" is a member of "PERSONNEL RECORD." All data elements of a structure are identified in this fashion.

1. Class Structure Representation

Class structures are represented by parenthesized expressions which are used to define the structure of the classificatory tree.

The technique of employing parentheses to define structures is similar to that technique employed in LISP S-expressions [Ref. 12]. Punctuation symbols used in the expressions are the left parenthesis, the right parenthesis, and the comma. The parentheses are used to enclose those classes which are directly related to a superclass while the comma is used to separate the classes within the parenthesized unit. Units within an expression are separated by commas and the entire expression itself is enclosed by parentheses. As demonstrated in the preceding section, "PERSONNEL RECORD" consists of the classes: "NAME," "ADDRESS," and "TELEPHONE NUMBER." This definition is called the format definition and is the foundation for the construction of the classificatory tree. Format definitions are represented by the parenthesized expression shown below.

PERSONNEL RECORD (NAME, ADDRESS, TELEPHONE NUMBER)

"NAME" and "ADDRESS" were further divided into subclasses and the expressions below show the parenthesized forms for "class definitions."

NAME (LAST, FIRST, MIDDLE)

ADDRESS (STREET, CITY, STATE)

Subclasses may also be subdivided and this process is replicated to fully define the class structure of the universe of discourse. Figure 4 graphically demonstrates the class structuring process, the fully parenthesized expression for the class structure, and the associated classificatory tree. Although the above example does not include a

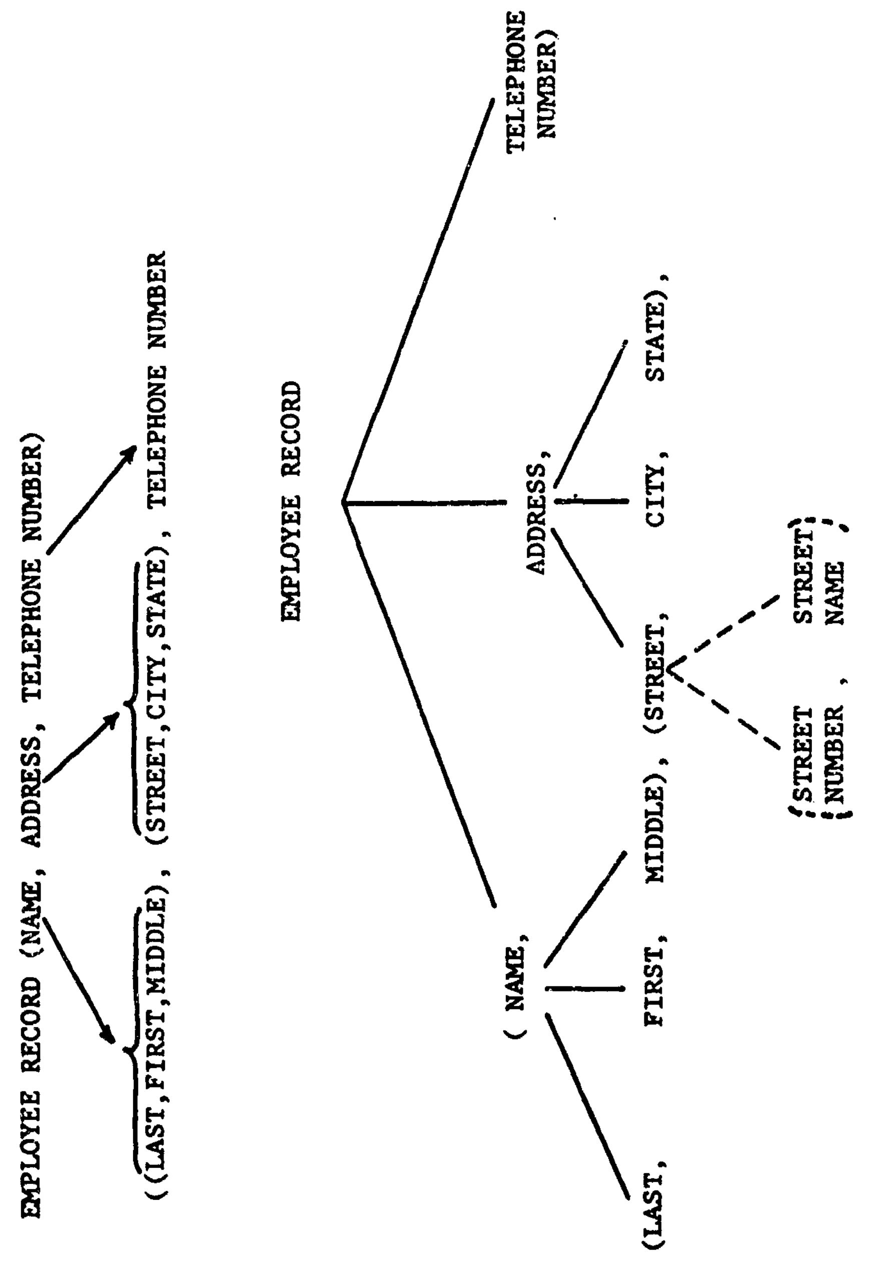


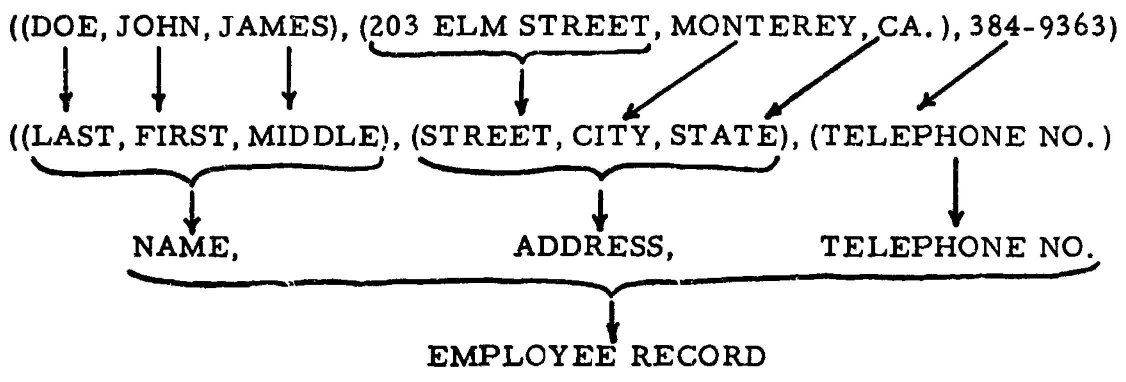
Figure 4

Parenthesized Class Expressions and Associated Free Structure for the Hierarchical Classification of Data.

subdivision for the class "STREET" one is shown in the tree structure to demonstrate a third level of class replication.

2. Data Representation

Once the class structure is defined, the associated data may be mapped directly onto the structure. Data representation is identical to the class expression as shown below.


((DOE, JOHN, JAMES), (203 ELM STREET, MONTEREY, CA.), 384-9363)
((LAST, FIRST, MIDDLE), (STREET, CITY, STATE), (TELEPHONE NO.)
NAME,
ADDRESS,
TELEPHONE NO.
EMPLOYEE RECORD

Representation of repeated data elements within the record are easily handled by properly parenthesizing the record. For example, two phone numbers for John Doe would be represented by:

((DOE, JOHN, JAMES), (203 ELM STREET, MONTEREY, CAL.),
(384-9363, 384-6214))

The class membership of each data element in the record is clearly defined by the parenthesized expression.

3. System Utility

The utility of hierarchical classification in association with parenthesized expressions is realized by the user in three ways:

1. The indexing techniques presented in section II require the user to conform to the language devised by the system designer for the retrieval of information. The user does not have the option of defining the indexing language that best suits his particular needs but must be satisfied with the indexing technique employed to best satisfy the needs of all users. In contrast, this system allows each user or user group to define his own indexing language by defining the class structure associated with the data he is most concerned with. In other words, the system will accept a mix of data allowing each user or user group to have his own retrieval system within a retrieval system. Each user or user group must define the class structure of his data. For example, a business-oriented system might consist of a data base partitioned into employee records, pay records, stock inventory, etc. Such a system would simultaneously serve the needs of many users.

2. The user has the capability of entering the data structure in several ways to extract desired information. In the personnel record example, the user may retrieve complete records which satisfy certain search keys, or retrieve only the names of personnel, or retrieve the phone number of a particular person, and so on.

3. The classification scheme could serve as an intermediate language between the query processor and the retrieval system.

B. RETRIEVAL PROCESS

1. Query Format

Queries are presented to the system utilizing the same format as class expressions. The fully parenthesized expression contains search keys and blank positions which specify the information to be supplied to the user. The retrieval processor will fill in the blank positions with all of the information contained in the data base which satisfies the search keys. The expression must conform identically to the fully parenthesized expression used to represent the class structure.

((DOE, JOHN, ___), (____, ____ , ____), ___) In the example above, the system will identify the class membership of each search key and blank position through the classificatory tree constructed from the class expression. A search is then instituted for all records which contain an occurrence of "DOE" as a member of the class "LAST" and "JOHN" as a member of the class "FIRST." Information is extracted from those appropriate records to fill the blank positions of the query. The user may broaden or narrow the amount of information retrieved by the number and/or class of search keys used in the query. A query containing only the search key "CALIFORNIA" could produce a greater amount of information than a query which has only one blank position.

2. Boolean Expressions

The ability to use Boolean expressions such as "and," "or," "not," etc., is desirable in any information retrieval package. However, the degree to which Boolean expressions may be used is left to the perogative of the system designer in satisfying user needs. The use of Boolean "and" is accepted by the retrieval processor in this system and is identified by the amphersand:

((___, ___, ___), (___, MONTEREY & MARINA, CALIFORNIA), ___)

In this case, the names, street addresses, and phone numbers of all personnel who live in Monterey, California <u>and</u> Marina, California would be produced.

The use of Boolean "or" is not directly used in this system but its effect is similar to the use of alphabetic and numeric range requests.

3. Alphabetic and Numeric Ranges

Alphabetic and numeric range requests are identified by the colon. Examples of range requests are exhibited below.

((A : D, —, —), (—, MONTEREY, CALIFORNIA), —)

The retrieval processor identifies an alphabetic range request for all data elements which are members of the class "LAST" and which have as a first letter A, B, C, or D. The records of all personnel who live in Monterey, California and whose last names begin with A through D inclusive would be produced.

As shown immediately above, the system does not restrict the use of alphabetic or numeric ranges to single letters but any number of characters may be used and any number of range requests are possible within a single query.

The above discussion is also true for numeric range requests. For example, the user desires complete records for all those personnel who have specific telephone exchanges:

((— , — , —), (— , — , —), 372:394)

VII. SYSTEM STRUCTURE

This section discusses the internal design of the gener .. -purpose fact-retrieval system employing the data-structure technique previously explained. The system was i nplemented on the Naval Postgraduate School's IBM 360 Model 67 Computer and is an interactive system under control of the Cambridge Monitor System (CP/CMS) [Ref. 13].

A. DATA FILES

Data files are stored on punched cards and consist of the following three types:

1. Format definition car'`. These cards define the class structure for each universe of discourse to be included in the data base. An example of a format definition card is:

EMPLOYEE RECORD (NAME, ADDRESS, AGE, CHILDREN)

2. Class definition cards. These cards further define the structure of the classes contained in the for:nat definition. Examples of class definition cards are:

NAME (LAST, FIRST)

ADDRESS (STREET, CITY, STATE)

3. Data records. The data records contain the data elements associated with the universe of discourse and are fully parenthesized expressions. An example of a data record is:

EMPLOYEE RECORD ((DOE, JOHN), (203 ELM STREET, MONTEREY, CA.), (48), (MARY, SALLY))

Format definitions, class definitions, and data records may also be entered into the system via on-line terminal. For a large-scale data base, the data records could be stored in unstructured form on a back-up storage device such as magnetic tape. Structuring of records would be accomplished under program control according to pre-stored format and class definitions.

B. TREE-TYPE DATA STRUCTURES

A tree-type data structure is employed to represent the hierarchical classification of a universe of discourse. The tree-structuring process described later in this section employs data cells to represent nodes within a tree and the "chaining" technique to order the cells into tree structure form.

1. Data Cells

Data cells available to the tree-structuring processor consist of three fields. The description and function of each field is described below:

a. The identifier field, referred to as "TOP," contains the storage address (pointer) of the data or class entity which the data cell represents.

b. The right link field, referred to as "RIGHT," contains a pointer which is used to chain the data cell to another data cell on the same level of the tree.

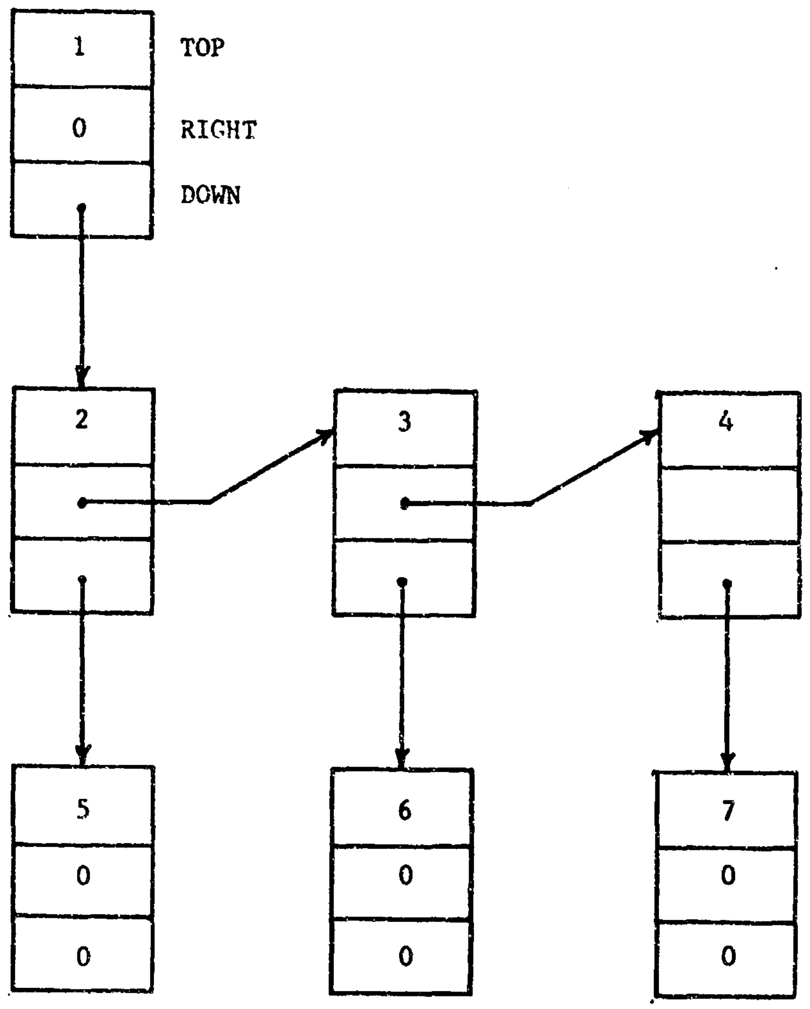
c. The down link field, referred to as "DOWN," contains a pointer which is used to chain the data cell to another data cell located in a lower level of the tree. Figure 5 demonstrates the use of data cells. A zero in a link field signifies "no link" or a null field.

2. Structuring Process

Empty data cells are constructed in core storage through list structuring techniques and are stored in an area available to the tree-structuring routine. The reading of a format definition card initiates the structuring process. The format name (e.g., EMPLOYEE RECORD) and the class names contained on the card are extracted and moved into storage (a discussion of this process is deferred to a later section). A number of cells equal to the format name plus the number of class names contained on the card are retrieved and tree structuring commences. The first cell in the tree structure is called a "header" and serves to identify the format name of the tree. Each of the classes contained in the format definition is assigned to a data cell and the cells are chained together. Figure 6 shows the structure representing the format definition:

EMPLOYEE RECORD (NAME, ADDRESS, AGE, CHILDREN)

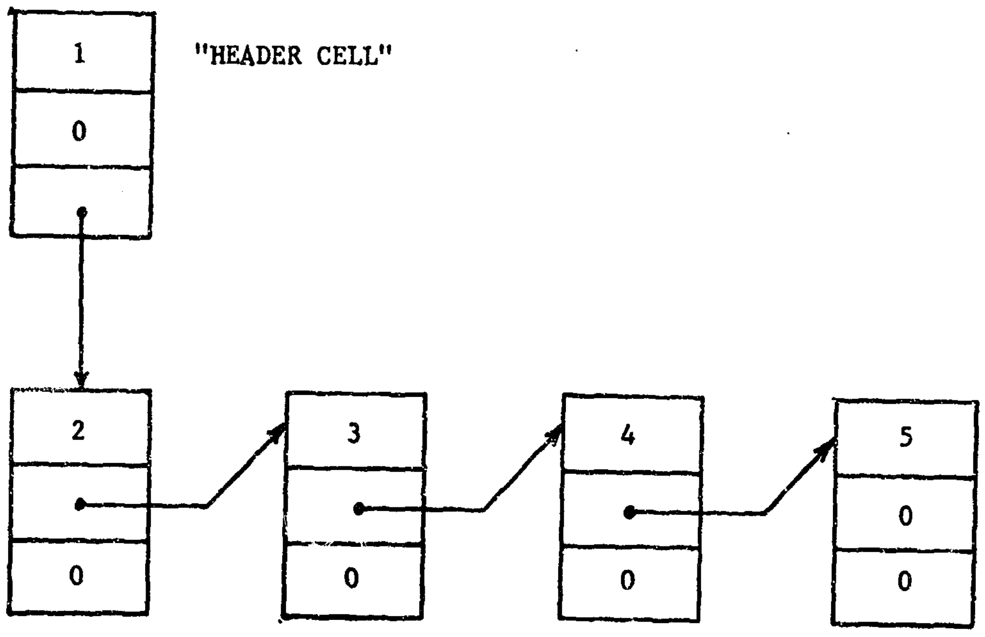
Before completing the discussion of tree structuring it is important to note that class definitions throughout the various universes of discourse in the data base must be consistent. That is to say, if the class called "NAME" is defined as (LAST, FIRST) then every occurrence of "NAME" must consist of the classes "LAST" and "FIRST." If this is not done, confusion arises during the retrieval process when



NOTE: The numbers in the TOP fields are sequence numbers.

Figure 5

Data Cell Composition



EMPLOYEE RECORD (NAME, ADDRESS, AGE, CHILDREN)

The numbers in the TOP fields correspond to:

1	EMPLOYEE RECORD
2	NAME
3	ADDRESS
4	AGE
5	CHILDREN

Figure 6

Tree Structure Composed of Data Cells

the processor attempts to identify the class memberships of data elements. Therefore, as each format definition is read, a search is conducted of all previously constructed trees to determine whether or not each of the classes contained in the definition being processed have been previously used. If a class has been previously used then the tree structure representing the class is appended to the tree being built. If a class has not been previously used then a class definition card must be submitted to the tree-structuring processor.

After the format definition card has been processed any class definition cards associated with the structure are processed. Figure 7 contains a completed tree structure for:

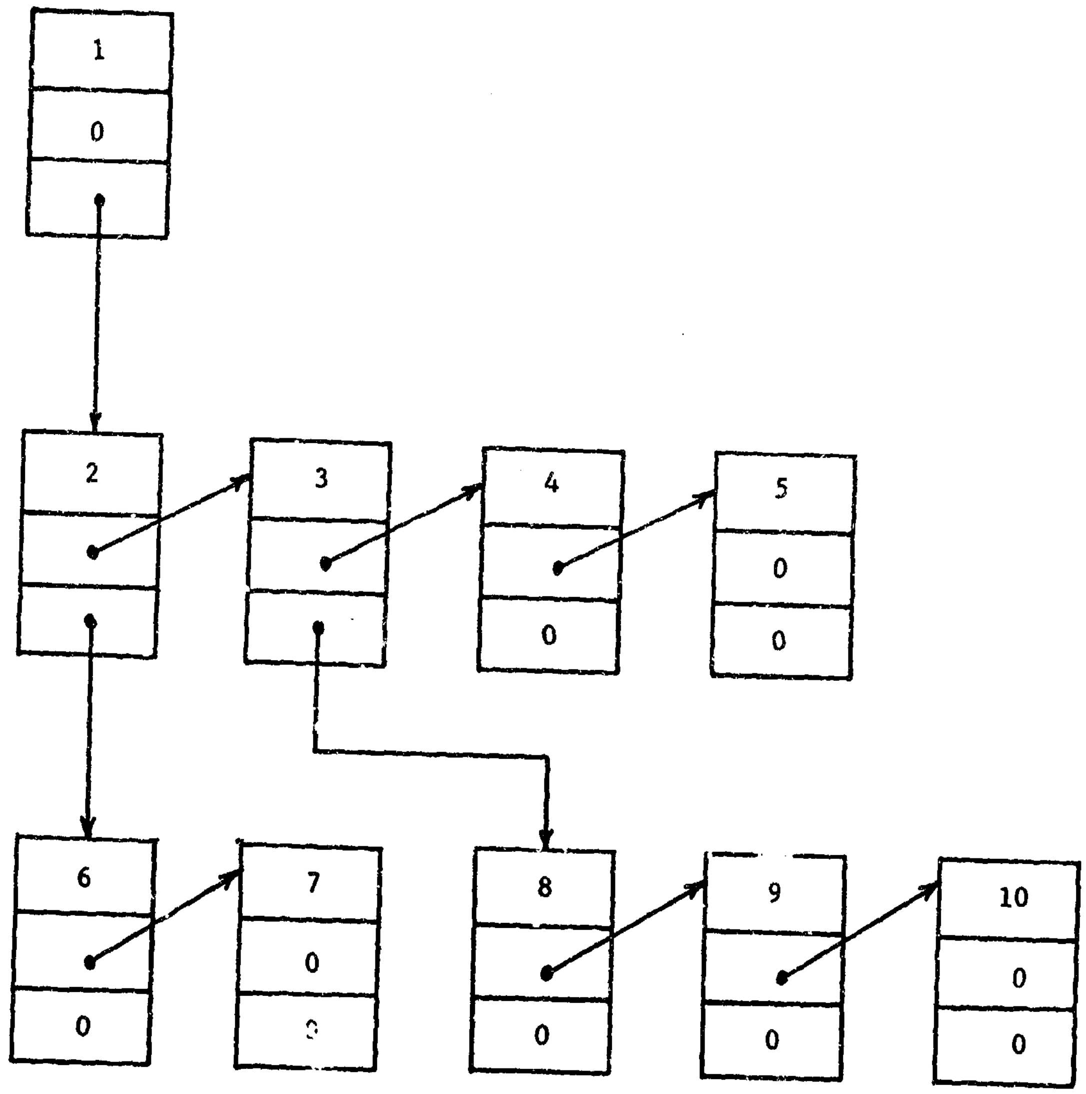
EMPLOYEE RECORD (NAME, ADDRESS, AGE, CHILDREN)

NAME (LAST, FIRST)

ADDRESS (STREET, CITY, STATE)

C. INDEX FILES

The system incorporates an index file, called the master index, which demonstrates many of the characteristics and advantages of an inverted index. The master index contains format names, class names, and data elements. Each entry in the index has a pointer associated with it which links the entry to a tree structure, data record, or further information concerning the entry. The retrieval process is always initiated at the master index since it is the agent which directs the search for information in response to a user's query.



EMPLOYEE RECORD (NAME, ADDRESS, AGE, CHILDREN)

NAME (LAST, FIRST)

ADDRESS (STREET, CITY, STATE)

The numbers in the TOP fields correspond to:

1	EMPLOYEE RECORD	6	LAST
2	NAME	7	FIRST
3	ADDRESS	8	STREET
4	AGE	9	CITY
5	CHILDREN	10	STATE

Figure 7

Tree Structure for the Format: "EMPLOYEE RECORD"

1. Characteristics of the Master Index

Conceptually, the master index is a large matrix consisting of fixed-length records (matrix rows), each containing eight fields (matrix columns), as shown in Figure 8. The first four characters of format names, class names, and data elements are stored in the first four fields of the index. Entries which contain more than four characters are then stored in a sequential storage area reserved for variable-length records. The remaining four fields of each index record contain information concerning the type of entry (e.g., format name, class name, or data element), the sequential store address of the full character representation of the entry, if any, pointers to information-bearing data cells, and other information useful to the retrieval processor.

2. Constructing the Master Index

The first record of the master index is reserved as a table of all format names contained in the data base. The first record contains the address of the first data cell (identical to the data cells used in tree structuring) in a chain of cells and each cell contains the address of a format name located in sequential storage. Through this record a user may quickly determine the partitioning of the data base. Figure 9 demonstrates the idea.

Format names are entered in the index and linked to their definitions which are located in sequential storage. Each of the classes contained in the format definitions are also stored in the index.

	1	2	3	4	5	6	7	8
1	C	L	A	S		1		1
2	E	M	P	L	1	7	22	1
3	N	A	M	E	C		69	106
4	A	D	D	R	C	51	88	37
5	A	G	E		L	0	55	10
6	C	H	I	L	L	59	61	13
7	L	A	S	T	L	0	7	124
8	F	I	R	S	L	82	13	127
9	S	T	R	E	L	108	19	130
10	C	I	T	Y	L	0	25	133
11	S	T	A	T	L	115	31	55
12	D	Ø	E		D	0		4
13	J	Ø	H	N	D	0		10
14	2	0	3	E	D	194		16
⋮	⋮				⋮			⋮

<u>COLUMN(S)</u>

1-4 : First 4 characters of the entry

5 : No. if the entry is a format name
"C" if the entry is a class name
"L" if the entry is the lowest level class in a tree structure
"D" if the entry is a data element

6 : Pointer to the full character representation in sequential store

7 : Pointer to associated chain of data cells if the entry is classified "L", otherwise pointer to sequential store

8 : Pointer to associated data cell in the tree structure if the entry is a class or format.
Pointer to associated chain of data cells if the entry is a data element.

Figure 8

Representation of the Master Index

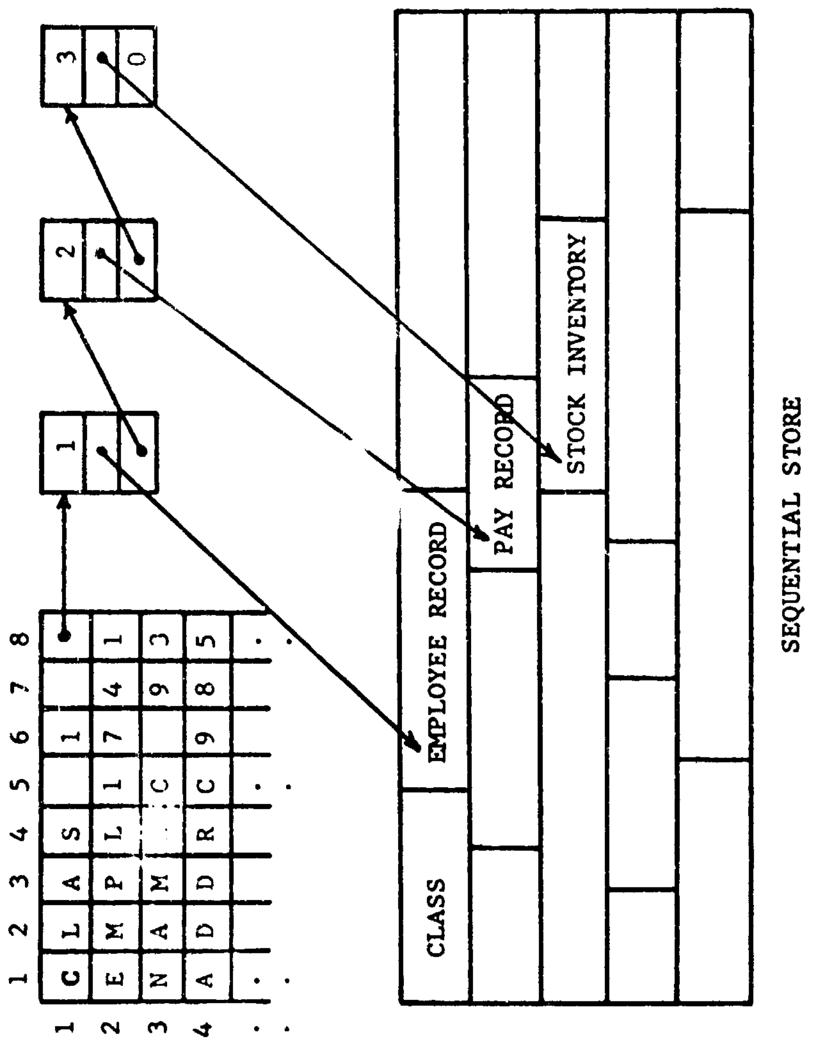


Figure 9

Reserved Record in the Master Index for Format Names with Associated Data Cells and Forma. Names in Sequential Store

Associated with each class entry in the index is a string of data cells which contain two items of information concerning the class:

a. The first field contains the number of the data record which, in turn, contains an occurrence of the class. (This information is added when the data records are read and is discussed later.)

b. The second field contains a number corresponding to the format name which contains this class entry.

A class may be used in any number of different format definitions but its structure must be consistent in every occurrence. Therefore, regardless of the number of format definitions which contain a given class, there is only one index record for the class. The data cells appended to the class entry provide the retrieval processor with data such as the format definitions in which the class appears. Among other things, information pertaining to the class entries provides the retrieval processor with the capability of quickly abandoning a search when a user requests information through a class which is not a member of the format being queried.

Class definitions are processed in a manner very similar to format definition processing. The class being defined is entered in the index and the definition is stored as read in the sequential store. The system returns the sequential store address and enters it in the index record. Appropriate data cells are appended to the index and the class structure is added to the classificatory tree. When the tree is completed, those classes which are end nodes in the classificatory tree (e.g., LAST, FIRST, STREET, CITY, STATE, AGE, and

CHILDREN in EMPLOYEE RECORD) are identified and their index records are flagged. This is done to ensure that elements in the data records are mapped onto the tree structure according to their proper class membership.

As each data record is read into the system it is assigned a unique number and placed in the sequential store. Each element within the record is examined to determine its class membership and the master index is searched to determine if the element was previously entered by another data record. The possibility of a data element appearing in more than one record exists if the data base contains similar formats such as employee records and pay records. In addition, a data element may be a member of more than one class such as the occurrence of "JOHN" as a member of both classes "FIRST" and "CHILDREN." It is highly desirable that there be only one entry in the master index for those elements which occur more than once. Unique entries in the index guarantees that when an item is located in the index, the search process is complete and successful. Additionally, the need for combined search plans is eliminated. Specific record and class membership information for each data element entered in the index is resolved by appending data cells to the master index entry. The data cells contain the record number(s) from which the element was extracted and its class membership(s). Assuming that a data element occurs several times in the data base, the master index would still contain only one record for the element. The record contains all of the information pertinent to the retrieval process. The technique, relevant to both class and data entries, results in two important savings:

1. A significant reduction of storage space is realized (if an element occurs several times) since multiple entries in the master index require more storage space than a single record and its associated data cells.

2. A significant reduction in search time is realized since multiple entries require the retrieval processor to conduct a full-file search each time it enters the master index.

3. <u>Data Record Table</u>

Cells appended to each data element stored in the master index do not contain the sequential store addresses of the records from which the data elements were extracted. This information is stored separately in a table referred to as a data record table. The data record table augments the information contained in the master index and is composed of fixed-length records as shown in Figure 10. Each table record consists of three fields which contain:

a. The unique data record number.

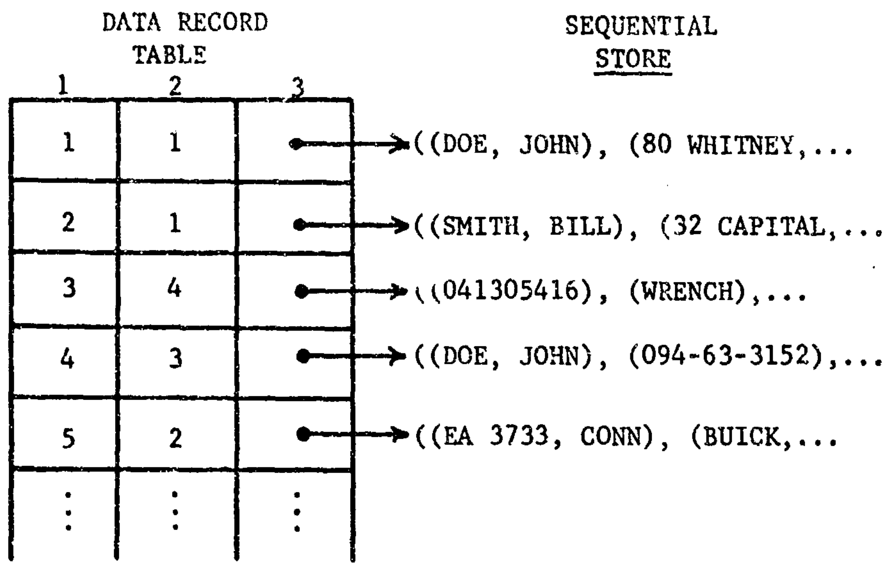
b. Format membership of the data record.

c. Sequential store address of the data record.

The data record table serves two functions:

a. The retrieval processor bypasses the master index and directly enters the data record table to satisfy requests for all data records which are members of a particular universe of discourse.

b. The table is also utilized for queries other than those which request "all data records." The retrieval processor searches the master index to determine the data records which satisfy a user's


DATA RECORD TABLE
SEQUENTIAL STORE
1 2 3
1 1
2 1
3 4
4 3
5 2
((DOE, JOHN), (80 WHITNEY,...
((SMITH, BILL), (32 CAPITAL,...
((041305416), (WRENCH),...
((DOE, JOHN), (094-63-3152),...
((EA 3733, CONN), (BUICK,...

FORMAT NUMBER	FORMAT NAME
1	EMPLOYEE RECORD
2	CAR REGISTRATION
3	PAY RECORD
4	STOCK INVENTORY

COLUMN

1 :	Unique record number
2 :	Format membership of the data record
3 :	Pointer to data record in sequential store

Figure 10

Representation of the Data Record Table

request. Then the processor enters the data record table and extracts the sequential store addresses of the records. The sequential store addresses are passed to the "output" section of the retrieval processor.

The information contained in the data record table is tabulated separately from the master index to achieve savings in storage space and response time. Storage savings are realized since the addresses of data records in the sequential store are contained only in the data record table and are not replicated in the master index for each class and data element. System response time is reduced for queries that request all data records of a particular universe of discourse since the data record table was designed primarily to expedite this type of request. The retrieval processor extracts all of the necessary data record addresses in one access of the table. The amount of searching within the table is minimal.

D. INFORMATION FILE

The "sequential store" is the system information file, or data base. It contains the data records, format definitions, class definitions, and the full character representation of those entries in the master index consisting of more than four characters. Figure 11 shows the sequential store and its relationship to the master index and the data record table.

The information file is resident in main core storage. The variable-length records of this file are sequentially ordered. System information files are not normally stored in main core unless they are

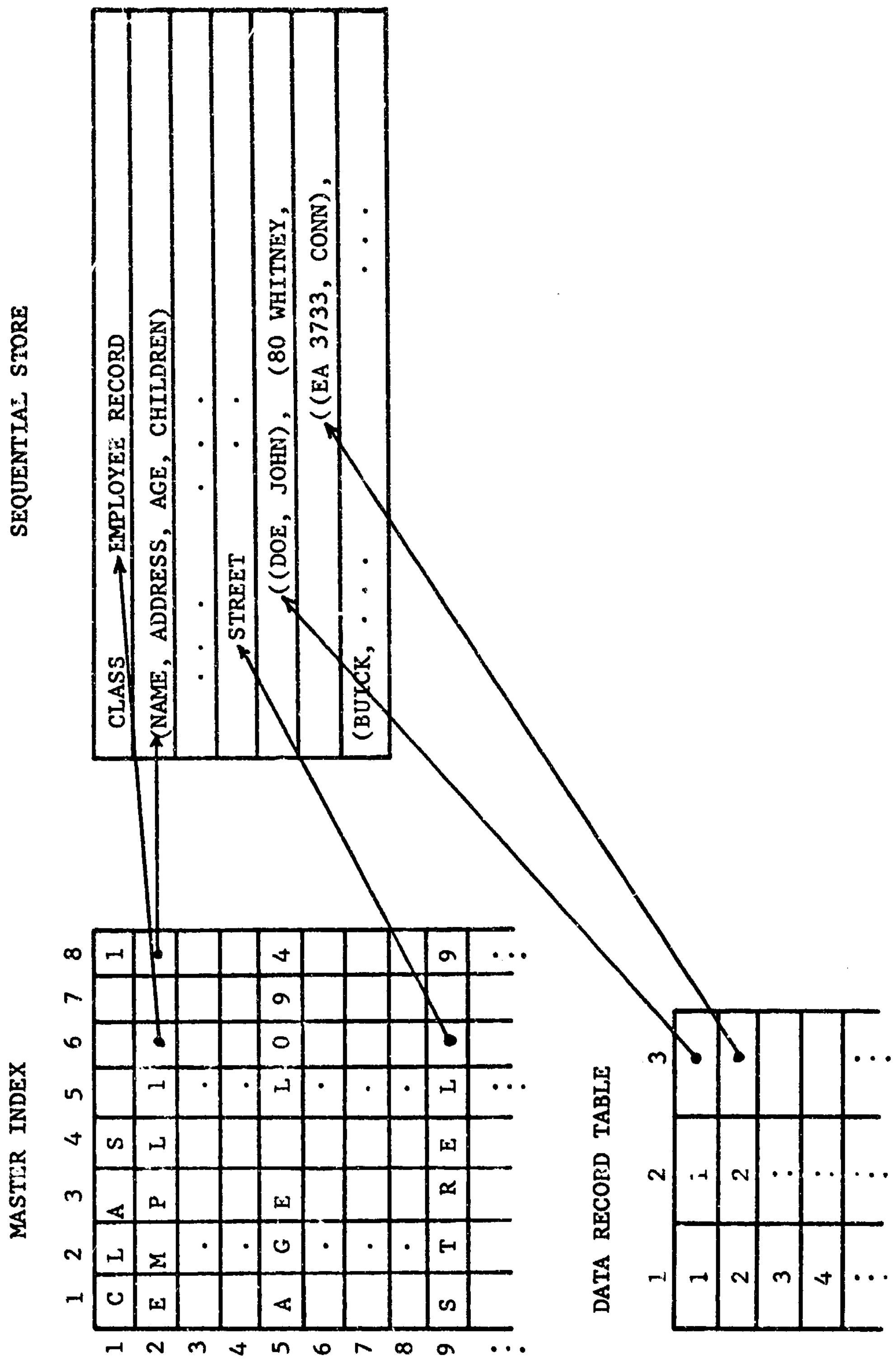


Figure 11

Relationship between the Master Index, Data Record Table, and Sequential Store relatively small (which is the case here). However, it is imperative that such a file be resident on a direct access storage device in order to provide satisfactory system response time.

E. RETRIEVAL PROCESSOR

The retrieval processor is divided into three operations. The identification operation determines the type of query posed by the user; the search operation determines the data record numbers which satisfy the user's request; the output operation retrieves the resultant data records from the sequential store and prints them at the terminal. Additionally, special messages are output to the user in the form of error messages to warn him of invalid queries, and messages which notify him of unsatisfied queries.

1. Query Types

The IR system designer strives to achieve total utility of the system by providing the user with a powerful retrieval language. Utility of the data structure used in this system is realized by the various types of queries available to the user for extracting information from the data base. There are four major types of queries available to the user.

a. Determining Data Base Partitions.

As previously discussed, the data base may be partitioned to allow a mix of unrelated information by defining the class structure of each universe of discourse in the data base. A user who is unfamiliar with the data base partitions (format names) may easily determine this information by submitting a special type of query. The format of the query is simple and consists of the single search key: "CLASS." This is translated by the retrieval processor as: "Output the names of all formats contained in the data base." Search of the master index is then centered at the first record of the index and its associated chain of data cells which contain the sequential store addresses of the format names. All format names contained in the data base are output to the user.

```
     QUERY:   CLASS
  RESPONSE:   EMPLOYEE RECORD
              PAY RECORD
                .
                .
                .
```

b. Determining Format and Class Definitions.

In order to extract data from a specific universe of discourse, the user must be provided with its class structure. The class structure determines the format for data record requests. Queries of format and class definitions must contain, as a search key, the format name or class name to be defined. The search processor enters the master index to locate the format name or class name, extracts the address of its definition located in the sequential store, and the definition is output directly at the terminal.

QUERY:	EMPLOYEE RECORD
RESPONSE:	(NAME, ADDRESS, AGE, CHILDREN)
QUERY:	NAME
RESPONSE:	(LAST, FIRST)
QUERY:	AGE
RESPONSE:	NO DESCENDANTS
.	.
.	.
.	.

c. Data Element R rieval.

One asset of the data structure concept is that it allows the user to extract single data elements from the data base which are members of a particular class and format, or members of a particular class irrespective of the format membership. Since data elements are mapped onto the end nodes of their respective tree structures, the user must use the lowest level classes of the structure as search keys. Failure to do so prompts the retrieval processor to output corrective information to the user. The hyphens in the queries below indicate to the retrieval processor that the expressions are queries and not format definitions. The processor could identify the expression by searching the master index for an occurrence of "EMPLOYEE RECORD." A successful search would indicate that a format definition already existed in the system. However, use of the hyphen is a simpler and faster method for positively identifying the type of expression submitted to the system.

```
        QUERY:    EMPLOYEE RECORD (NAME, __ )

     RESPONSE:    INVALID QUERY:
                  DETERMINE DESCENDANTS OF:  NAME
                  USE DESCENDANTS AS KEYWORDS


        QUERY:    EMPLOYEE RECORD (LAST, __ )

     RESPONSE:    BROWN

                  SMITH

                  THOMPSON
```

To answer the above query, a search is conducted in the master index for all data elements which are members of the class "LAST" and are members of the format "EMPLOYEE RECORD." This information is contained in the data cells appended to each data entry in the index. Elements which satisfy the query are taken directly from the master index, and output at the terminal.

In the query below, the hyphen is used to differentiate between a query and a class definition statement. All data elements which are members of the class "LAST" are output irrespective of format membership. The format membership fields of the data cells are ignored during the search of the master index.

```
        QUERY:      LAST (___)

     RESPONSE:      BROWN
                    CHAMBERS
                    COTTLE
                    DOE
                    SMITH
                    THOMPSON
```

d. Data Record Retrieval.

Data record retrieval is the most valuable and would be the most frequently used type of request available to the system user. Extraction of complete data records which satisfy the search keys contained in the query is accomplished. To retrieve data records, the queries contain data elements as search keys and may contain Boolean "AND," alphabetic and/or numeric ranges, or any combination thereof. The query format is a fully parenthesized expression as shown in previous sections. Search keys are positioned in the expression with respect to class membership and hyphens inserted in those positions for which information is requested. Any variation from the properly parenthesized expression prompts error messages from the retrieval processor to the user.

The retrieval process for the query listed below is explained in the following paragraphs:

EMPLOYEE RECORD ((DOE, —), (—, —, CA.)(—), (—))

The mat name appearing at the beginning of the query expression informs the retrieval processor of the universe of discourse in which the user is interested. The processor then traverses the tree structure for "EMPLOYEE RECORD" to determine the lowest level classes in the tree. This information, in conjunction with the proper use of parentheses in the query expression, allows the processor to identify the class memberships of the search keys contained in the query. The user is notified whenever the processor is unable to find a search key in the master index. In this case, the processor attempts to recover data which satisfies the remaining search keys. Similar action takes place when the processor encounters a search key which is not a member of the class specified in the query, or if a search key is not a member of the format specified in the query. Additionally, the user is notified whenever the query is improperly formatted.

Each search key in the query is processed sequentially. The retrieval processor searches the master index for an occurrence of each key. Record numbers which contain an occurrence of the search key are extracted and stored in a list. After all search keys have been processed, the retrieval processor "ANDS" the record numbers in the list to determine which records satisfy the query. For example, assuming that two key words are used and record numbers 5, 32, and 67 satisfy the first key word, and record numbers 32 and 67 satisfy the second key word, records 32 and 67 are output to the user. Record numbers which satisfy the query are passed to the "output" section of the retrieval processor which retrieves the sequential store addresses of the records from the data record table and prints the records at the terminal.

A user has the ability to immediately examine the results of his query since the system is interactive. The results of one query may prompt the user to submit another request, either broadening or narrowing the request through judicious use of search keys. In any case, the user is guaranteed that if the information that he seeks is contained in the data base, he will have quick and easy access to it. Appendix A contains a sample run of the fact-retrieval system and demonstrates all of the queries available to a user and the system responses.

F. ALTERING THE DATA BASE

1. Changes and Deletions

Due to the experimental nature of the system, no utility routines have been provided for deleting records or making changes to existing records. Alterations are accomplished by manually changing the card images in the data files.

2. Additions

The addition of data records to existing data sets or the submission of new universes of discourse are accomplished most easily without special utility routines. This feature is inherently built into the system through the data structuring technique. Addition of a new universe of discourse is accomplished by submitting format and class definitions, and associated data records either on-line through the terminal (automatically) or off-line with card images (manually). New data records may also be added to existing data files automatically or manually.

VIII. CONCLUSIONS

Characteristics of the data-structuring concept as used in a general-purpose fact-retrieval system have been discussed throughout the preceeding sections. These concepts are summarized here.

The data structuring technique encompasses the concept of hierarchical classification which is the most widely used method of indexing. Hierarchical classification of data is a relatively simple technique to use but possesses the power to divide and subdivide a universe of discourse into more specific subjects. Additionally, hierarchical structures may be created to include a domain of subjects. This is advantageous for use in a fact-retrieval system, as previously demonstrated, by providing a mix of structures in a single data base. Therefore, users with differing interests are provided simultaneous access to a single system since each is provided a "personal" retrieval system within a larger retrieval system. In addition, the hierarchical structure provides a user with multiple avenues of access into his information file.

Parenthesized expressions serve as an intermediate language between the query processor and the information retrieval system. The query processor is able to determine the class memberships of elements within an expression by examination of the parenthesized form. It is apparent, however, that the use of parenthesized expressions is cumbersome and demanding since misplacing parentheses is easy to do and causes loss of meaning of the expression. On the other hand, it can be argued that the technique of parenthesizing expressions is powerful and an equally powerful substitute is difficult to theorize.

Request all format names (data base partitions).

```
class*
FORMAT NUMBER  1  EMPLOYEERECORD
FORMAT NUMBER  2  PAYRECORD
FORMAT NUMBER  3  CST8/SOCIAL
FORMAT NUMBER  4  61/SOCIAL

REQUEST COMPLETE
```

Request specific format and class definitions.

```
employeerecord*
EMPLOYEERECORD(NAME,ADDRESS,AGE,CHILDNAME)


name*
NAME(LAST,FIRST)


address*
ADDRESS(STREET,CITY,STATE)


age*
AGE                             HAS NO DESCENDANTS


childname*
CHILDNAME                       HAS NO DESCENDANTS



cst8/social*
CST8/SOCIAL(FNAME,FADDRESS,DOR,WIFE)


fname*
FNAME(LAST,FIRST,MI,RANK)
```

Request all data elements which are members of a specific class.

```
cst8/social(fname,-)*
INVALID QUERY:
DETERMINE DESCENDANTS OF:  FNAME
USE DESCENDANTS AS KEYWORDS


fname*
FNAME(LAST,FIRST,MI,RANK)


cst8/social(last,-)*

COLLINS

KOTTKE

REQUEST COMPLETE
```

Request all data records which satisfy specific keywords.

```
employeerecord((-,doe),(-,-,-),-,-)*
DOE                                 WAS FOUND BUT IS NOT A MEMBER OF
THE CLASS SPECIFIED IN THE QUERY:
RECORDS SATISFYING OTHER KEYWORDS,IF ANY,ARE LISTED
REQUEST NOT FULFILLED: NO RECORDS SATISFY THE QUERY


employeerecord((doe,-),(-,-,-),-,-)*


((DOE,JOHN),((203ELMSTREET,MTRY,CA.),(80WHITNEYAVE.,BPT,CONN.)),48,MARY)

REQUEST COMPLETE


name((doe,-),(-,-,-),-,-)*
INVALID QUERY:
NAME                                 IS NOT A FORMAT NAME


doe((-,john),(-,-,-),-,-)*
INVALID QUERY:
DOE                                  IS A DATA ELEMENT
```

```
cst8/social((-,-,-,lt),(-,-,-,-),(-,65:68),-)*

((KOTTKE,ROBERT,A.,LT),(3370SANLUISAVE.,CARMEL,6240461,2504),(DEC,66),JAN)

REQUEST COMPLETE

cst8/social((colins,james,-,-),(-,-,-,-),(-,-),-)*
COLINS                          WAS NOT FOUND:
RECORDS SATISFYING OTHER KEYWORDS,IF ANY,ARE LISTED


((COLLINS,JAMES,E.,LCDR),(918LOSPALOSDRIVE,SALINAS,4247041,3235),(JAN,68),BETTY)

REQUEST COMPLETE

cst8/social((collins,james,-,-),(-,-,-,-),(-,-),-)*


((COLLINS,JAMES,E. LCDR),(918LOSPALOSDRIVE,SALINAS,4247041,3235),(JAN,68),BETTY)

REQUEST COMPLETE

cst8/social((kottke&petrucci,-,-,-),(-,-,-,-),(-,-),-)*
PETRUCCI                        WAS FOUND BUT IS NOT A MEMBER OF
THE FORMAT SPECIFIED IN THE QUERY:
IT IS A MEMBER OF:
EMPLOYEERECORD
RECORDS SATISFYING OTHER KEYWORDS,
IF ANY,ARE LISTED


((KOTTKE,ROBERT,A. LT),(3370SANLUISAVE.,CARMEL,6240461,2504),(DEC,66),JAN)

REQUEST COMPLETE

cst8/social(-)*

((COLLINS,JAMES,E.,LCDR),(918LOSPALOSDRIVE,SALINAS,4247041,3235),(JAN,68),BETTY)

((KOTTKE,ROBERT,A.,LT),(3370SANLUISAVE.,CARMEL,6240461,2504),(DEC,66),JAN)

REQUEST COMPLETE
```

```
employeerecord((doe,richard),(-,mtry,-),-,-)*
REQUEST NOT FULFILLED: NO RECORDS SATISFY THE QUERY


employeerecord((doe,-),(-,mtry,-),-,-)*


((DOE,JOHN),((203ELMSTREET,MTRY,CA.),(80WHITNEYAVE.,BPT,CONN.)),48,MARY)

REQUEST COMPLETE


employeerecord((pizinger,richard),(-,-,-),-,-)*
PIZINGER                         WAS NOT FOUND:
RECORDS SATISFYING OTHER KEYWORDS,IF ANY,ARE LISTED


((PETRUCCI,RICHARD),(489FORESTCIRCLE,MARINA,CA.),32,((DEIDRE),(KRISTEN)))

REQUEST COMPLETE


employeerecord((doe&petrucci,-),(-,-,-),-,-)*


((DOE,JOHN),((203ELMSTREET,MTRY,CA.),(80WHITNEYAVE.,BPT,CONN.)),48,MARY)


((PETRUCCI,RICHARD),(489FORESTCIRCLE,MARINA,CA.),32,((DEIDRE),(KRISTEN)))

REQUEST COMPLETE


employeerecord((,-,-),(-,-,-),50:55,-)*
INVALID QUERY: NUMBER OF KEYWORD
POSITIONS EXCEEDS THE NUMBER OF CLASSES
CONTAINED IN THE SPECIFIED FORMAT


employeerecord((-,-),(-,-,-),50:55,-)*
REQUE T NOT FULFILLED: NO RECORDS SATISFY THE QUERY
```

```
employeerecord((-,-),(-,-,-),40:55,-)*

((DOE,JOHN),((203ELMSTREET,MTRY,CA.),(80WHITNEYAVE.,BPT,CONN.)),48,MARY)

REQUEST COMPLETE

employeerecord((d:p,-),(-,-,-),-,-)*

((DOE,JOHN),((203ELMSTREET,MTRY,CA.),(80WHITNEYAVE.,BPT,CONN.)),48,MARY)

( 'PETRUCCI,RICHARD),(489FORESTCIRCLE,MARINA,CA.),32,((DEIDRE),(KRISTEN ))

REQUEST COMPLETE
```

Addition of a new format name (data set) to the data base.

```
car registration(registry,car,name,address)*


registry(number,state)*


car(year,make,model)*


car registration((ea3733,conn),(66,buick,skylark),(smith,  william),
(1132 hayer st.,westport,conn))*


car registration ((hr4738,conn),(60,vw,sedan),(graham ,carl),
(4 bauer pl.,stamford,conn))*

car registration((hv973,conn),(65,buick,riviera),(kelly,louis),
(1433 post rd.,fairfield,conn))*
```

Queries pertaining to the new format name.

```
car registration*
CARREGISTRATION(REGISTRY,CAR,NAME,ADDRESS)

registry*
REGISTRY(NUMBER,STATE)

name*
NAME(LAST,FIRST)

address*
ADDRESS(STREET,CITY,STATE)

car registration(-)*

((EA3733,CONN),(66,BUICK,SKYLARK),(SMITH,WILLIAM),(1132BAYERST.,WESTPORT,CONN))

( WR4738,CONN),(60,VW,SEDAN),(GRAHAM,CARL),(4BAUERPL.,STAMFORD,CONN))

((HV978,CONN),(65,BUICK,RIVIERA),(KELLY,LOUIS),(1433POSTRD.,FAIRFIELD,CONN))

REQUEST COMPLETE

car registration((-,conn),(65:70,-,-),(-,-),(-,-,-))*

((EA3733,CONN),(66,BUICK,SKYLARK),(SMITH,WILLIAM),(1132BAYERST.,WESTPORT,CONN))

REQUEST COMPLETE
```

```
      IMPLICIT INTEGER*2 (A-Z)
      DIMENSION FTAB(30)
      DIMENSION UPRAN(30)
      DIMENSION RECTAB(100)
      DIMENSION TOP(603),RIGHT(602),DOWN(601)
      DIMENSION WORK(241),ACUM(241),FRST4(241)
      DIMENSION LEVEL(50),DRTAB(50,3),DESTAB(30),NODTAB(50)
      DIMENSION RNO(603),CMEM(602),NEXT(601)
      DIMENSION SERCH(30),MINDEX(500,8),SEQ(4000)
      COMMON/ONE/TOP,AVAIL1,P,Q,RR,DD,YY
      COMMON/TWO/RNO,AVAIL2,S,R,CM,RM
      COMMON/THREE/ANS,SERCH,MINDEX,ACUM,SJ,SK,N,SEQ
      COMMON/FOUR/NODTAB,FD
      COMMON/FIVE/MULT,MK,EK
      COMMON/SIX/S2,AT,AE,HCTR,UPRAN
      EQUIVALENCE (TOP(3),RIGHT(2),DOWN(1))
      EQUIVALENCE (RNO(3),CMEM(2),NEXT(1))
      DATA OP/'('/,CP/')'/,BLANK/' '/,COMMA/','/,STAR/'*'/,
     1EQS/'='/,CX/'C'/,DX/'D'/,DOLS/'$'/
      DATA LX/'L'/,AX/'A'/,SX/'S'/,COL/':'/,
     1AMP/'&'/,HYP/'-'/
C-----INITIALIZE ARRAYS,CERTAIN COUNTERS AND SUBSCRIPTS.
      AVAIL1=1
      Q=AVAIL1
      AVAIL2=1
      S=AVAIL2
      R=S
      CALL INIT1
      CALL INIT2
      TERM=0
      ER=0
      FT=0
      MI=1
      FNUM=0
      RNUM=0
      DO 20 I=1,50
      DO 21 J=1,3
      DRTAB(I,J)=BLANK
   21 CONTINUE
   20 CONTINUE
      DO 23 I=1,500
      MINDEX(I,7)=BLANK
      DO 24 J=1,4
      MINDEX(I,J)=BLANK
   24 CONTINUE
   23 CONTINUE
      DO 26 I=1,30
      FTAB(I)=BLANK
   26 DESTAB(I)=BLANK
C-----RESERVE THE FIRST ROW OF THE MASTER INDEX FOR'CLASS'.
C-----MINDEX(1,8) POINTS TO CELLS WHICH CONTAIN FORMAT
C-----NUMBERS,AND POINTERS TO THE FULL CHARACTER REPRESENTA-
C-----TION OF THE FORMAT NAME IN SEQUENTIAL STORAGE. CELLS
C-----ARE ATTACHED AS THE RECORD FORMATS ARE PROCESSED.
      MINDEX(1,1)=CX
      MINDEX(1,2)=LX
      MINDEX(1,3)=AX
      MINDEX(1,4)=SX
      MINDEX(1,5)=0
      MINDEX(1,6)=1
      MINDEX(1,8)=0
      SEQ(1)=CX
      SEQ(2)=LX
      SEQ(3)=AX
      SEQ(4)=SX
      SEQ(5)[illegible]X
      SEQ(6)=STAR
      SI=6
   44 DO 28 I=1,30
   28 SERCH(I)=BLANK
   45 J=1
      K=80
      PCTR=0
      DO 27 I=1,241
      WORK(I)=BLANK
   27 ACUM(I)=BLANK
      IF(TERM.EQ.1.OR.ER.EQ.1) GO TO 399
C-----READ ONE RECORD INTO THE 'WORK' ARRAY AND DETERMINE
C-----IF THE PARENTHESES ARE BALANCED OR IF THE RECORD
C-----EXCEEDS 240 CHARACTERS.
   46 READ (4,2,END=900) (WORK(I),I=J,K)
    2 FORMAT (80A1)
      GO TO 41
  399 READ (5,2) (WORK(I),I=J,K)
      IF(WORK(1).EQ.DOLS.OR.WORK(2).EQ.DOLS.OR.WORK(3)
     1.EQ.DOLS) GO TO 1000
   41 DO 30 L=J,K
      IF(WORK(L).EQ.OP) PCTR=PCTR+1
      IF(WORK(L).EQ.CP) PCTR=PCTR-1
      IF(WORK(L+1).EQ.STAR) GO TO 32
   30 CONTINUE
      J=J+80
      K=K+80
      IF(K.GT.240) GO TO 950
      IF(TERM.EQ.1.OR.ER.EQ.1) GO TO 399
      GO TO 46
   32 IF(PCTR) 925,40,925
C-----DEBLANK THE RECORD CONTAINED IN WORK, LOAD IT INTO THE
C-----ACCUMULATOR, AND DETERMINE ITS LENGTH.
   40 ER=0
      J=0
      DO 47 I=1,241
      IF(WORK(I).EQ.BLANK) GO TO 47
      J=J+1
      ACUM(J)=WORK(I)
      IF(WORK(I).EQ.STAR) GO TO 48
   47 CONTINUE
   48 N=J-1
C-----DETERMINE THE RECORD TYPE AND BRANCH TO THE
C-----APPROPRIATE BLOCK OF CODE FOR PROCESSING.
C
C
      CALL IDENT(&600,&800,&700)
C
C
C-----THIS BLOCK OF CODE PROCESSES INPUT RECORDS,
C-----I.E.,FORMAT DEFINITIONS,CLASS DEFINITIONS,AND DATA
C-----RECORDS. FORMAT OR CLASS TREES ARE STRUCTURED,ENTRIES
C-----MADE IN THE MASTER INDEX,DATA RECORD TABLE,AND
C-----SEQUENTIAL STORAGE.
      IF(ACUM(1).EQ.EQS) GO TO 251
      DO 50 I=1,N
      MK=I
      IF(ACUM(I).EQ.OP) GO TO 55
   50 SERCH(I)=ACUM(I)
   55 IF (ACUM(MK+1).NE.CP) GO TO 56
      WRITE(6,280)
  280 FORMAT(1H ,'INVALID QUERY: MISSING HYPHEN')
      GO TO 44
   56 NN=N
      N=MK-1
      CALL MISRCH
      IF(ANS.EQ.0) GO TO 57
      FNO=MINDEX(SJ,5)
      GO TO 200
C----THIS RECORD IS A FORMAT.
C----ADD A CELL TO MINDEX(1,8) FOR THIS FORMAT.
   57 FNUM=FNUM+1
      IF(R.NE.S) NEXT(R)=0
      CALL GET2
      NEXT(R)=0
      IF(MINDEX(1,8).NE.0) GO TO 71
      MINDEX(1,8)=R
      CM=R
      GO TO 73
   71 CM=MINDEX(1,8)
   72 IF(NEXT(CM).EQ.0) GO TO 78
      CM=NEXT(CM)
      GO TO 72
   78 NEXT(CM)=R
      CM=NEXT(CM)
   73 RNO(CM)=FNUM
      SI=SI+1
      CMEM(CM)=SI
      MI=1
   74 MI=MI+1
      IF(MINDEX(MI,1).NE.BLANK) GO TO 74
      LK=MK-1
      DO 77 I=1,LK
      IF(I.GT.4) GO TO 60
      MINDEX(MI,I)=ACUM(I)
   60 SEQ(SI)=ACUM(I)
      SI=SI+1
   77 CONTINUE
   79 SEQ(SI)=STAR
      SI=SI+1
C-----SET A POINTER TO THE FORMAT DEFINITION STORED IN 'SEQ'
      MINDEX(MI,7)=SI
      DO 82 L=MK,NN
      SEQ(SI)=ACUM(L)
      SI=SI+1
   82 CONTINUE
      SEQ(SI)=STAR
C-----SET A POINTER TO THE FORMAT NAME STORED IN'SEQ'.
      MINDEX(MI,6)=CMEM(CM)
      MINDEX(MI,5)=FNUM
C-----INITIALIZE A HEADER CELL FOR THE FORMAT TREE.
      CALL GET1
      AVAIL1=P
      MINDEX(MI,8)=P
      TOP(P)=MI
C-----CHAIN A CELL TO THE HEADER FOR THE FIRST CLASS OF THIS
C-----FORMAT DEFINITION.
      CALL GET1
      RR=P
      DD=P
      DOWN(P)=0
C-----DETERMINE IF EACH CLASS HAS BEEN PREVIOUSLY DEFINED
C-----EACH CLASS IS UNIQUELY DEFINED,THEREFORE,THERE WILL BE
C-----NO DUPLICATE CLASS ENTRIES IN THE MASTER INDEX.
C-----IF A CLASS HAS BEEN PREVIOUSLY DEFINED, ITS
C-----DESCENDANTS ARE LOCATED AND ADDED TO THE TREE.
      DI=0
      DJ=0
   85 DO 80 IK=1,30
      SERCH(IK)=BLANK
   80 FRST4(IK)=BLANK
   81 MK=MK+1
      IF(ACUM(MK).EQ.STAR) GO TO 170
      IF(ACUM(MK).EQ.COMMA.OR.ACUM(MK).EQ.OP) GO TO 81
      I=MK
      J=1
   90 SERCH(J)=ACUM(I)
      IF(J.GT.4) GO TO 95
      FRST4(J)=ACUM(I)
   95 I=I+1
      IF(ACUM(I).EQ.COMMA.OR.ACUM(I).EQ.CP) GO TO 100
      J=J+1
      GO TO 90
  100 N=J
      CALL MISRCH
      IF(ANS.EQ.1) GO TO 150
      MI=1
  105 MI=MI+1
      IF(MINDEX(MI,1).NE.BLANK) GO TO 105
      DO 110 JK=1,4
  110 MINDEX(MI,JK)=FRST4(JK)
      IF(J.GT.4) GO TO 120
      MINDEX(MI,6)=0
      GO TO 130
  120 SI=SI+1
      MINDEX(MI,6)=SI
      DO 125 K=1,J
      SEQ(SI)=SERCH(K)
  125 SI=SI+1
      SEQ(SI)=STAR
  130 MINDEX(MI,5)=CX
      IF(TOP(RR).EQ.0) GO TO 131
      CALL GET1
      RIGHT(RR)=P
      RR=RIGHT(RR)
  131 TOP(RR)=MI
      MINDEX(MI,8)=RR
      DOWN(RR)=0
C-----PROCESS THE NEXT CLASS IN THE FORMAT DEFINITION.
C-----IF NONE THEN READ IN THE NEXT RECORD.
      MK=I
      GO TO 85
  150 IF(MINDEX(SJ,5).EQ.LX) GO TO 151
      DI=DI+1
      DESTAB(DI)=SJ
      MI=SJ
  151 IF(TOP(RR).EQ.0) GO TO 155
      CALL GET1
      RIGHT(RR)=P
      RR=RIGHT(RR)
  155 TOP(RR)=SJ
      MINDEX(SJ,8)=RR
      DOWN(RR)=0
      MK=I
      GO TO 85
C-----ADD THE PREVIOUSLY DEFINED CLASSES TO THE TREE.
  170 MK=0
      IF(DESTAB(1).NE.BLANK) GO TO 175
  171 DO 173 I=1,30
      DESTAB(I)=BLANK
  173 CONTINUE
      GO TO 45
  175 DJ=DJ+1
      IF(DESTAB(DJ).EQ.BLANK) GO TO 171
  180 L=MINDEX(DESTAB(DJ),7)
      DO 193 J=1,241
  193 ACUM(J)=BLANK
      J=1
  197 ACUM(J)=SEQ(L)
      L=L+1
      J=J+1
      IF(SEQ(L).NE.STAR) GO TO 197
      ACUM(J)=SEQ(L)
      RR=MINDEX(DESTAB(DJ),8)
      CALL GET1
      DOWN(RR)=P
      RR=DOWN(RR)
      GO TO 85
C-----THIS BLOCK OF CODE PROCESSES CLASS DEFINITIONS
  200 IF(MINDEX(SJ,5).NE.CX.AND.MINDEX(SJ,5).NE.LX) GO TO 25
      SI=SI+1
      MINDEX(SJ,7)=SI
      DO 215 L=MK,NN
      SEQ(SI)=ACUM(L)
  215 SI=SI+1
      SEQ(SI)=STAR
      RR=MINDEX(SJ,8)
      CALL GET1
      DOWN(RR)=P
      RR=DOWN(RR)
      DOWN(RR)=0
      GO TO 85
```

```
C-----THE ACCUMULATOR CONTAINS A DATA RECORD.TRAVERSE THE
C-----TREE AND LOCATE ALL END NODES OF ALL BRANCHES.DATA
C-----ELEMENTS ARE MAPPED ONTO THEIR RESPECTIVE CLASSES.
C-----IF THE FIRST CHARACTER OF THE RECORD IS AN ASTERISK
C-----THEN THE RECORD IS A DATA RECORD AND A MEMBER OF THE
C-----SAME FORMAT AS THE LAST DATA RECORD PROCESSED.
  250 CALL TRAV
  251 NI=1
      NN=J-1
      RNUM=RNUM+1
C-----STORE THE RECORD IN SEQUENTIAL STORAGE AND MAKE DATA
C-----RECORD TABLE ENTRIES.
      SI=SI+1
      DRTAB(RNUM,1)=RNUM
      DRTAB(RNUM,2)=FNO
      DRTAB(RNUM,3)=SI
      IF(ACUM(1).EQ.EQS) MK=2
      DO 260 I=MK,NN
      SEQ(SI)=ACUM(I)
  260 SI=SI+1
      SEQ(SI)=STAR
C-----PROCESS EACH DATA ELEMENT IN THE RECORD.
      SET=0
      CALL CHAR4(ACUM,FRST4,NN)
C-----DETERMINE IF A MULTIPLE ENTRY EXISTS,
C-----E.G.,((JOHN DOE),(ROBERT SMITH))
      MK=MK+1
  255 IF(ACUM(MK).EQ.OP) CALL MENT
      IF(MULT.EQ.1) MM=NI
  265 DO 267 I=1,30
  267 SERCH(I)=BLANK
      J=1
  270 SERCH(J)=ACUM(MK)
      J=J+1
      MK=MK+1
  271 IF(ACUM(MK).EQ.STAR) GO TO 44
      IF(ACUM(MK).EQ.COMMA.OR.ACUM(MK).EQ.CP) GO TO 275
      GO TO 270
C-----DETERMINE IF THE DATA ELEMENT HAS BEEN PREVIOUSLY
C-----ENTERED IN THE MASTER INDEX. IF NOT MAKE ENTRIES IN
C-----THE MASTER INDEX,SEQUENTIAL STORE,AND INITIALIZE THE C
C-----MEMBERSHIP CELLS.
  275 N=J-1
      CALL MISRCH
      IF(ANS.EQ.0) GO TO 535
      SET=SET+4
      DO 455 JJ=1,30
  455 SERCH(JJ)=BLANK
      CM=MINDEX(SJ,8)
  460 IF(NEXT(CM).EQ.0) GO TO 465
      CM=NEXT(CM)
      GO TO 460
  465 NEXT(R)=[illegible]
      CALL GET2
      NEXT(CM)=R
      RNO(R)=RNUM
      CMEM(R)=NODTAB(NI)
      SJ=NODTAB(NI)
      NI=NI+1
      IF(NODTAB(NI).NE.BLANK) GO TO 480
      GO TO 45
  480 IF(MINDEX(SJ,7).EQ.BLANK) GO TO 493
      CM=MINDEX(SJ,7)
  485 IF(NEXT(CM).EQ.0) GO TO 490
      CM=NEXT(CM)
      GO TO 485
  490 NEXT(R)=0
      CALL GET2
      NEXT(CM)=R
      GO TO 495
  493 NEXT(R)=0
      CALL GET2
      MINDEX(SJ,7)=R
  495 RNO(R)=RNUM
      CMEM(R)=FNO
  492 IF(ACUM(MK).NE.COMMA) GO TO 498
  497 MK=MK+1
      IF(ACUM(MK).EQ.OP) GO TO 255
      GO TO 265
  498 MK=MK+1
      IF(ACUM(MK).EQ.STAR) GO TO 44
      IF(ACUM(MK).EQ.COMMA.OR.ACUM(MK).EQ.CP) GO TO 498
      IF(ACUM(MK).EQ.OP) GO TO 500
      GO TO 265
  500 IF(MULT.NE.1) GO TO 255
      MULT=0
      NI=MM
  505 MK=MK+1
      IF(ACUM(MK).EQ.STAR) GO TO 44
      IF(ACUM(MK).EQ.OP) GO TO 505
      GO TO 265
  535 DO 540 M=1,4
      SET=SET+1
  540 MINDEX(SJ,M)=FRST4(SET)
      IF(J.LE.5) GO TO 555
      SI=SI+1
      MINDEX(SJ,6)=SI
      LL=1
  545 SEQ(SI)=SERCH(LL)
      LL=LL+1
      IF(LL.EQ.J) GO TO 550
      SI=SI+1
      GO TO 545
  550 SI=SI+1
      SEQ(SI)=STAR
  555 IF(J.LE.5) MINDEX(SJ,6)=0
      MINDEX(SJ,5)=DX
      NEXT(R)=0
      CALL GET2
      MINDEX(SJ,8)=R
      RNO(R)=RNUM
      CMEM(R)=NODTAB(NI)
      L=NODTAB(NI)
      NI=NI+1
      IF(MINDEX(L,7).EQ.BLANK) GO TO 580
      CM=MINDEX(L,7)
  565 IF(NEXT(CM).EQ.0) GO TO 570
      CM=NEXT(CM)
      GO TO 565
  570 NEXT(R)=0
      CALL GET2
      NEXT(CM)=R
      RNO(R)=RNUM
      CMEM(R)=FNO
      GO TO 492
  580 NEXT(R)=0
      CALL GET2
      MINDEX(L,7)=R
      RNO(R)=RNUM
      CMEM(R)=FNO
      GO TO 492
C-----THIS BLOCK OF CODE PROCESSES FORMAT DEFINITION,CLASS
C-----DEFINITION,AND FORMAT NAME QUERIES.
  600 DO 605 I=1,N
  605 SERCH(I)=ACUM(I)
      CALL MISRCH
      IF(ANS.EQ.1) GO TO 615
  607 WRITE(6,610) (SERCH(I),I=1,30)
  610 FORMAT(1H ,'REQUEST NOT FULFILLED:',
     1  /,T2,30A1,'WAS NOT FOUND')
      GO TO 44
  615 IF(SJ.NE.1) GO TO 645
C-----OUTPUT ALL FORMAT NAMES.
      CM=MINDEX(1,8)
  618 ST=CMEM(CM)
      SE=ST
  620 IF(SEQ(SE+1).EQ.STAR) GO TO 625
      SE=SE+1
      GO TO 620
  625 WRITE (6,630) RNO(CM),(SEQ(I),I=ST,SE)
  630 FORMAT(1H ,'FORMAT NUMBER',I3,2X,30A1)
      IF(NEXT(CM).EQ.0) GO TO 635
      CM=NEXT(CM)
      GO TO 618
  635 WRITE (6,640)
  640 FORMAT (/T2,'REQUEST COMPLETE')
      FT=0
      GO TO 44
  645 IF(MINDEX(SJ,5).NE.DX) GO TO 650
  646 WRITE(6,647) (SERCH(I),I=1,30)
  647 FORMAT(1H ,'INVALID QUERY:',
     1/,T2,30A1,'IS A DATA ELEMENT')
      GO TO 44
  650 IF(MINDEX(SJ,5).NE.CX.AND.MINDEX(SJ,5).NE.LX)
     1    GO TO 670
      IF(MINDEX(SJ,5).NE.LX) GO TO 652
      WRITE(6,653) (SERCH(I),I=1,30)
  653 FORMAT(1H ,30A1,'HAS NO DESCENDANTS')
      GO TO 44
  652 ST=MINDEX(SJ,7)
      SE=ST
  655 IF(SEQ(SE+1).EQ.STAR) GO TO 660
      SE=SE+1
      GO TO 655
  660 WRITE(6,665) (ACUM(I),I=1,N),(SEQ(IX),IX=ST,SE)
  665 FORMAT(1H ,30A1,3(80A1,/))
      GO TO 44
  670 DO 673 I=1,N
  673 SERCH(I)=ACUM(I)
      CALL MISRCH
      IF(ANS.NE.1) GO TO 607
      GO TO 652
C-----IF THE KEYWORD SPECIFIED IN THE QUERY IS A FORMAT NAME
C-----THEN OUTPUT ALL DATA RECORDS WHICH ARE MEMBERS OF THAT
C-----FORMAT.IF THE KEYWORD IS A CLASS THEN OUTPUT ALL DATA
C-----ELEMENTS WHICH ARE MEMBERS OF THAT CLASS.
  700 DO 705 I=1,30
      IF(ACUM(I).EQ.OP) GO TO 707
      N=I
      SERCH(I)=ACUM(I)
  705 CONTINUE
  707 CALL MISRCH
      IF(ANS.EQ.1) GO TO 709
      WRITE(6,610) (SERCH(I),I=1,30)
      GO TO 44
  709 IF(MINDEX(SJ,5).EQ.DX) GO TO 646
  710 IF(MINDEX(SJ,5).EQ.CX.OR.MINDEX(SJ,5).EQ.LX) GO TO 750
  720 J=0
  725 J=J+1
      IF(DRTAB(J,2).EQ.BLANK) GO TO 635
      IF(DRTAB(J,2).NE.MINDEX(SJ,5)) GO TO 725
      ST=DRTAB(J,3)
      SE=ST
  730 IF(SEQ(SE+1).EQ.STAR) GO TO 733
      SE=SE+1
      GO TO 730
  733 WRITE (6,735) (SEQ(I),I=ST,SE)
  735 FORMAT (2(/,T2,120A1))
      GO TO 725
  750 IF(MINDEX(SJ,5).EQ.LX) GO TO 760
      WRITE(6,753) (SERCH(I),I=1,30)
  753 FORMAT(1H ,'INVALID QUERY:',
     1/,T2,'DETERMINE DESCENDANTS OF:  ',30A1,
     2/,T2,'USE DESCENDANTS AS KEYWORDS')
      GO TO 44
  760 MI=1
  765 MI=MI+1
      IF(MINDEX(MI,1).EQ.BLANK) GO TO 635
      IF(MINDEX(MI,5).NE.DX) GO TO 765
      CM=MINDEX(MI,8)
  770 IF(CMEM(CM).EQ.SJ) GO TO 771
  773 IF(NEXT(CM).EQ.0) GO TO 765
      CM=NEXT(CM)
      GO TO 770
C-----IF FT=1 THEN OUTPUT ONLY THOSE DATA ELEMENTS WHICH
C-----ARE MEMBERS OF THE CLASS AND FORMAT SPECIFIED
C-----IN THE QUERY.
  771 IF(FT.EQ.0) GO TO 774
      IF(DRTAB(RNO(CM),2).NE.FNO) GO TO 773
  774 IF(MINDEX(MI,6).EQ.0) GO TO 785
      ST=MINDEX(MI,6)
      SE=ST
  778 IF(SEQ(SE+1).EQ.STAR) GO TO 780
      SE=SE+1
      GO TO 778
  780 WRITE (6,783) (SEQ(I),I=ST,SE)
  783 FORMAT (/,T2,30A1)
      GO TO 765
  785 WRITE (6,788) (MINDEX(MI,I),I=1,4)
  788 FORMAT (/,T2,4A1)
      GO TO 765
C-----THIS BLOCK OF CODE PROCESSES HYPHEN AND
C-----BOOLEAN'AND' REQUESTS.
  800 DO 801 I=1,100
      RECTAB(I)=BLANK
  801 CONTINUE
      RI=0
      RSMK=1
      DO 805 I=1,30
      IF(ACUM(I).EQ.OP) GO TO 810
      N=I
      SERCH(I)=ACUM(I)
  805 CONTINUE
  810 CALL MISRCH
      IF(ANS.EQ.0) GO TO 607
      IF(MINDEX(SJ,5).NE.CX.AND.MINDEX(SJ,5).NE.LX)
     1    GO TO 820
      WRITE(6,815) (SERCH(I),I=1,30)
  815 FORMAT(1H ,'INVALID QUERY:',
     1/,T2,30A1,'IS NOT A FORMAT NAME')
      GO TO 44
  820 IF(MINDEX(SJ,5).EQ.DX) GO TO 646
      FNO=MINDEX(SJ,5)
      CALL TRAV
      DO 821 I=1,50
      IF(NODTAB(I).EQ.BLANK) GO TO 823
      NODE=I
  821 CONTINUE
  823 CLAS=0
      DO 824 I=1,241
      IF(ACUM(I).EQ.COMMA) CLAS=CLAS+1
      IF(ACUM(I).EQ.STAR) GO TO 826
  824 CONTINUE
  826 IF(CLAS.LT.NODE) GO TO 827
      WRITE(6,828)
  828 FORMAT(1H ,'INVALID QUERY: NUMBER OF KEYWORD ',
     1/,T2,'POSITIONS EXCEEDS THE NUMBER OF CLASSES ',
     2/,T2,'CONTAINED IN THE SPECIFIED FORMAT')
      GO TO 44
  827 CALL QSCAN(&822,&855,&870)
C----CMNO IS CLASS MEMBERSHIP NO.
  822 CMNO=NODTAB(HCTR)
      DI=0
      CALL MISRCH
      IF(ANS.EQ.1) GO TO 831
      WRITE(6,825) (SERCH(I),I=1,30)
  825 FORMAT(1H ,30A1,2X,'WAS NOT FOUND:',
     1/,T2,'RECORDS SATISFYING OTHER KEYWORDS,IF ANY,',
    2  'ARE LISTED')
      AT=AE
      CALL QSCAN1(&822,&855,&870)
  830 IF(RECTAB(RI).EQ.BLANK.OR.RECTAB(RI).EQ.STAR)
    1  GO TO 893
      RI=RI+1
      RECTAB(RI)=STAR
      RSMK=RI
      AT=AE
      CALL QSCAN1(&822,&855,&870)
  831 IF(MINDEX(SJ,5).EQ.CX) GO TO 834
      IF(MINDEX(SJ,5).EQ.DX) GO TO 835
      FT=1
      GO TO 760
  834 WRITE(6,753) (SERCH(I),I=1,30)
      GO TO 44
  835 CM=MINDEX(SJ,8)
  840 IF(CMEM(CM).EQ.CMNO) GO TO 842
  841 IF(NEXT(CM).EQ.0.AND.DI.EQ.0) GO TO 847
      IF(NEXT(CM).EQ.0) GO TO 843
      CM=NEXT(CM)
      GO TO 840
  842 DI=RNO(CM)
      IF(DRTAB(DI,2).EQ.FNO) GO TO 850
      GO TO 841
  847 WRITE(6,848) (SERCH(I),I=1,30)
  848 FORMAT(1H ,30A1,'WAS FOUND BUT IS NOT A MEMBER OF',
    1/,T2,'THE CLASS SPECIFIED IN THE QUERY:',
    2/,T2,'RECORDS SATISFYING OTHER KEYWORDS,IF ANY,',
    3  'ARE LISTED')
      AT=AE
      CALL QSCAN1(&822,&855,&870)
  843 WRITE(6,851) (SERCH(I),I=1,30)
  851 FORMAT(1H ,30A1,'WAS FOUND BUT IS NOT A MEMBER OF',
    1/,T2,'THE FORMAT SPECIFIED IN THE QUERY:',
    2/,T2,'IT IS A MEMBER OF:')
      I=0
      CM=MINDEX(SJ,8)
  310 I=I+1
      FTAB(I)=DRTAB(RNO(CM),2)
  315 IF(NEXT(CM).EQ.0) GO TO 325
      CM=NEXT(CM)
      DO 320 FI=1,30
      IF(FTAB(FI).EQ.BLANK) GO TO 310
      IF(FTAB(FI).EQ.DRTAB(RNO(CM),2)) GO TO 315
  320 CONTINUE
  325 CM=MINDEX(1,8)
  327 DO 330 I=1,30
      IF(FTAB(I).EQ.BLANK) GO TO 333
      IF(FTAB(I).EQ.RNO(CM)) GO TO 335
  330 CONTINUE
  333 IF(NEXT(CM).EQ.0) GO TO 355
      CM=NEXT(CM)
      GO TO 327
  335 ST CMEM(CM)
      SE=ST
  340 IF(SEQ(SE+1).EQ.STAR) GO TO 345
      SE=SE+1
      GO TO 340
  345 WRITE(6,350) (SEQ(IX),IX=ST,SE)
  350 FORMAT(1H ,T2,30A1)
      GO TO 333
  355 DO 357 I=1,30
  357 FTAB(I)=BLANK
      WRITE(6,360)
  360 FORMAT(1H ,T2,'RECORDS SATISFYING OTHER KEYWORDS,',
    1/,T2,'IF ANY,ARE LISTED')
      AT=AE
      CALL QSCAN1(&822,&855,&870)
  850 DO 852 RK=RSMK,100
      IF(RECTAB(RK).EQ.BLANK) GO TO 853
      IF(RECTAB(RK).EQ.RNO(CM)) GO TO 854
  852 CONTINUE
  853 RI=RI+1
      RECTAB(RI)=RNO(CM)
  854 IF(S2.EQ.1) GO TO 362
      RI=RI+1
      RECTAB(RI)=STAR
      RSMK=RI
  362 AT=AE
      CALL QSCAN1(&822,&855,&870)
C-----THIS BLOCK OF CODE PROCESSES ALPHABETIC AND/OR
C-----NUMERIC RANGE REQUESTS.
  855 CMNO=NODTAB(HCTR)
      SJ=2
  857 IF(MINDEX(SJ,5).NE.DX) GO TO 865
      I=1
      IF(MINDEX(SJ,6).EQ.0) GO TO 860
      J=MINDEX(SJ,6)
  858 IF(SEQ(J).EQ.SERCH(I).AND.SEQ(J).EQ.UPRAN(I))
     1GO TO 859
      IF(SEQ(J).GE.SERCH(I).AND.SEQ(J).LE.UPRAN(I))
     1GO TO 861
      GO TO 865
  861 IF(SEQ(J).EQ.SERCH(I)) GO TO 863
      IF(SEQ(J).EQ.UPRAN(I)) GO TO 866
      GO TO 864
  859 I=I+1
      J=J+1
      IF(SERCH(I).EQ.BLANK) GO TO 864
      GO TO 858
  863 I=I+1
      J=J+1
      IF(SERCH(I).EQ.BLANK.OR.SEQ(J).GE.SERCH(I))
     1  GO TO 864
      GO TO 865
  866 I=I+1
      J=J+1
      IF(SERCH(I).EQ.BLANK.OR.SEQ(J).LE.[illegible]RAN(I))
     1  GO TO 864
      GO TO 865
  860 IF(MINDEX(SJ,I).EQ.SERCH(I).AND.M[illegible]X(SJ,I)
     1.EQ.UPRAN(I)) GO TO 862
      IF(MINDEX(SJ,I).GE.SERCH(I).AND.M[illegible]EX(SJ,I)
     1.LE.UPRAN(I)) GO TO 871
      GO TO 865
  871 IF(MINDEX(SJ,I).EQ.SERCH(I)) GO TO 873
      IF(MINDEX(SJ,I).EQ.UPRAN(I)) GO TO 876
      GO TO 864
  862 I=I+1
      IF(SERCH(I).EQ.BLANK) GO TO 864
      GO TO 860
  873 I=I+1
      IF(SERCH(I).EQ.BLANK.OR.MINDEX(SJ,I).GE.SERCH(I))
     1   GO TO 864
      GO TO 865
  876 I=I+1
      IF(SERCH(I).EQ.BLANK.OR.MINDEX(SJ,I).LE.UPRAN(I))
     1   GO TO 864
  865 SJ=SJ+1
      IF(MINDEX(SJ,1).NE.BLANK) GO TO 857
      S1=0
      GO TO 830
  864 CM=MINDEX(SJ,8)
  856 IF(CMEM(CM).EQ.CMNO) GO TO 868
  867 IF(NEXT(CM).EQ.0) GO TO 865
      CM=NEXT(CM)
      GO TO 856
  868 DI=RNO(CM)
      IF(DRTAB(DI,2).EQ.FNO) GO TO 872
      GO TO 867
  872 DO 883 RK=RSMK,100
      IF(RECTAB(RK).EQ.BLANK) GO TO 869
      IF(RECTAB(RK).EQ.RNO(CM)) GO TO 865
  883 CONTINUE
  869 RI=RI+1
      RECTAB(RI)=RNO(CM)
      GO TO 865
C-----RECORD NUMBERS WHICH SATISFY INDIVIDUAL QUERY KEYWORDS
C-----ARE STORED IN A LIST ('RECTAB').THIS BLOCK OF CODE
C-----SEARCHES 'RECTAB'TO DETERMINE WHICH RECORDS SATISFY
C-----ALL QUERY KEYWORDS.
  870 IF(RI.EQ.0) GO TO 893
      IF(RECTAB(RI).EQ.STAR) GO TO 887
      RI=RI+1
      RECTAB(RI)=STAR
  887 RECTAB(RI+1)=DOLS
      W=0
      RI=1
      TRNO=RECTAB(RI)
      RS=1
  880 RS=RS+1
      IF(RECTAB(RS).NE.STAR) GO TO 880
      RMK=RS
  881 RS=RS+1
  882 IF(RECTAB(RS).EQ.DOLS) GO TO 895
  885 IF(RECTAB(RS).EQ.TRNO) GO TO 890
      RS=RS+1
      IF(RECTAB(RS).NE.STAR) GO TO 885
      IF(W.EQ.0) GO TO 893
      GO TO 635
  890 RS=RS+1
      W=1
      IF(RECTAB(RS).EQ.STAR) GO TO 881
      GO TO 890
  893 WRITE(6,892)
  892 FORMAT(1H ,'REQUEST NOT FULFILLED: '
     1 'NO RECORDS SATISFY THE QUERY')
      GO TO 44
  895 ST=DRTAB(TRNO,3)
      SE=ST
  896 IF(SEQ(SE+1).EQ.STAR) GO TO 898
      SE=SE+1
      GO TO 896
  898 WRITE (6,899) (SEQ(I),I=ST,SE)
  899 FORMAT (/,T2,2(/,T2,120A1))
      RI=RI+1
      IF(RI.EQ.RMK) GO TO 635
      TRNO=RECTAB(RI)
      RS=RMK+1
      GO TO 882
  900 WRITE (6,10)
   10 FORMAT(1H0,T40,'* * * FLY NAVY * * *')
      TERM=1
      NEXT(R)=0
      GO TO 45
  925 WRITE(6,6)
    6 FORMAT(1H ,'ERROR: UNBALANCED PARENTHESIS')
      WRITE(6,7) (WORK(I),I=1,240)
    7 FORMAT(T2,2(/,T2,120A1))
      IF(TERM.EQ.1) GO TO 45
      ER=1
      GO TO 45
  950 WRITE(6,8)
    8 FORMAT(1H ,'ERROR: RECORD LENGTH EXCEEDS 240 CHARACTER
      WRITE(6,7) (WORK(I),I=1,240)
      IF(TERM.EQ.1) GO TO 45
      READ(4,2) (WORK(I),I=1,80)
      WRITE(6,2)
      ER=1
      GO TO 45
 1000 WRITE(6,999)
  999 FORMAT(T2,'PROGRAM TERMINATION')
      STOP
      END
      SUBROUTINE CHAR4(ARR,F4,L)
      IMPLICIT INTEGER*2 (A-Z)
      DIMENSION ARR(241),F4(241)
      DATA OP/'('/,CP/')'/,BLANK/' '/,COMMA/','/
C-----THIS SUBROUTINE STORES THE FIRST FOUR CHARACTERS OF
C-----EACH DATA ELEMENT IN THE ARRAY 'FRST4'.DURING DATA
C-----ELEMENT PROCESSING EACH CHARACTER BLOCK IS MOVED INTO
C-----THE MASTER INDEX,IF NOT PREVIOUSLY ENTERED.
      DO 5 I=1,241
    5 F4(I)=BLANK
      I=0
      J=1
      CCTR=0
    7 I=I+1
      IF(ARR(I).NE.OP) GO TO 7
   10 I=I+1
      IF(I.GE.L) GO TO 60
      IF(ARR(I).EQ.OP.OR.ARR(I).EQ.CP.OR.ARR(I).EQ.COMMA)
     1GO TO 20
   25 F4(J)=ARR(I)
      J=J+1
      CCTR=CCTR+1
      IF(CCTR-4) 10,30,30
   20 IF(ARR(I).EQ.OP.OR.CCTR.EQ.0) GO TO 10
   35 F4(J)=BLANK
      J=J+1
      CCTR=CCTR+1
      IF(CCTR-4) 35,30,30
   30 CCTR=0
      IF(ARR(I).NE.COMMA) GO TO 40
      I=I+1
      IF(ARR(I).NE.OP) GO TO 25
   40 I=I+1
      IF(I.GE.L) GO TO 60
      IF(ARR(I).NE.OP.AND.ARR(I).NE.CP.AND.ARR(I).NE.COMMA)
     1GO TO 40
   45 I=I+1
      IF(I.GE.L) GO TO 60
      IF(ARR(I).EQ.OP.OR.ARR(I).EQ.CP.OR.ARR(I).EQ.COMMA)
     1GO TO 45
      GO TO 25
   60 RETURN
      END

      SUBROUTINE INIT1
      IMPLICIT INTEGER*2 (A-Z)
      DIMENSION TOP(603),RIGHT(602),DOWN(601)
      COMMON/ONE/TOP,AVAIL1,P,Q,RR,DD,YY
      EQUIVALENCE (TOP(3),RIGHT(2),DOWN(1))
C-----THIS SUBROUTINE INITIALIZES CELLS USED IN TREE
C-----STRUCTURES.
      DO 10 I=1,601,3
      TOP(I)=0
   10 RIGHT(I)=0
      DO 20 I=1,598,3
   20 DOWN(I)=I+3
      DOWN(601)=0
      RETURN
      END

      SUBROUTINE GET1
      IMPLICIT INTEGER*2 (A-Z)
      DIMENSION TOP(603),RIGHT(602),DOWN(601)
      COMMON/ONE/TOP,AVAIL1,P,Q,RR,DD,YY
      EQUIVALENCE (TOP(3),RIGHT(2),DOWN(1))
      P=Q
      Q=DOWN(Q)
      RETURN
      END


      SUBROUTINE INIT2
      IMPLICIT INTEGER*2 (A-Z)
      DIMENSION RNO(603),CMEM(602),NEXT(601)
      COMMON/TWO/RNO,AVAIL2,S,R,CM,RM
      EQUIVALENCE (RNO(3),CMEM(2),NEXT(1))
C-----THIS SUBROUTINE INITIALIZES CLASS MEMBERSHIP CELLS.
      DO 10 I=1,601,3
      RNO(I)=0
   10 CMEM(I)=0
      DO 20 I=1,598,3
   20 NEXT(I)=I+3
      NEXT(601)=0
      RETURN
      END


      SUBROUTINE GET2
      IMPLICIT INTEGER*2 (A-Z)
      DIMENSION RNO(603),CMEM(602),NEXT(601)
      COMMON/TWO/RNO,AVAIL2,S,R,CM,RM
      EQUIVALENCE (RNO(3),CMEM(2),NEXT(1))
      R=S
      S=NEXT(S)
      RETURN
      END


      SUBROUTINE MISRCH
      IMPLICIT INTEGER*2 (A-Z)
      DIMENSION ACUM(241)
      DIMENSION SERCH(30),MINDEX(500,8),SEQ(4000)
      COMMON/THREE/ANS,SERCH,MINDEX,ACUM,SJ,SK,N,SEQ
      DATA BLANK/' '/,STAR/'*'/
C-----THIS SUBROUTINE SEARCHES THE MASTER INDEX FOR THE
C-----WORD CONTAINED IN THE ARRAY'SERCH'.
      ANS=0
      SJ=0
    5 SJ=SJ+1
      SK=1
      I=1
   10 IF(MINDEX(SJ,SK).EQ.SERCH(I)) GO TO 15
      SJ=SJ+1
      IF(MINDEX(SJ,SK).EQ.BLANK) GO TO 200
      GO TO 10
   15 IF(N-4) 16,16,20
   16 DO 17 J=2,4
      IF(MINDEX(SJ,J).NE.SERCH(J)) GO TO 5
   17 CONTINUE
      GO TO 100
   20 I=I+1
      KK=MINDEX(SJ,6)+1
      IF(SEQ(KK).NE.SERCH(I)) GO TO 5
   55 KK=KK+1
      IF(SEQ(KK).EQ.STAR) GO TO 90
      I=I+1
      IF(SEQ(KK).EQ.SERCH(I)) GO TO 55
      GO TO 5
   90 I=I+1
      IF(SERCH(I).NE.BLANK) GO TO 200
  100 ANS=1
  200 RETURN
      END
      SUBROUTINE IDENT(*,*,*)
      IMPLICIT INTEGER*2 (A-Z)
      DIMENSION SERCH(30),MINDEX(500,8),SEQ(4000),ACUM(241)
      COMMON/THREE/ANS,SERCH,MINDEX,ACUM,SJ,SK,N,SEQ
      DATA OP/'('/,CP/')'/,COMMA/','/,HYP/'-'/,AMP/'&'/,COL/
C-----THIS SUBROUTINE DETERMINES WHETHER THE ACCUMULATOR
C-----CONTAINS AN INPUT RECORD OR A QUERY AND RETURNS TO
C-----THE APPROPRIATE CODE BLOCK IN THE M/PROG FOR
C-----FURTHER TESTING AND PROCESSING.
      F1=0
      F2=0
      F3=0
      F4=0
      DO 10 I=1,N
      IF(ACUM(I).EQ.OP) F1=1
      IF(ACUM(I).EQ.HYP) F2=1
      IF(ACUM(I).EQ.COL) F3=1
      IF(ACUM(I).EQ.AMP) F4=1
   10 CONTINUE
   11 IF(F1.EQ.0) GO TO 40
      IF(F2.EQ.0.AND.F3.EQ.0.AND.F4.EQ.0) GO TO 30
      IF(F3.EQ.1.OR.F4.EQ.1) GO TO 50
      J=1
   15 IF(ACUM(J).EQ.OP) GO TO 17
      J=J+1
      GO TO 15
   17 DO 20 I=J,N
      IF(ACUM(I).EQ.OP.OR.ACUM(I).EQ.CP.OR.ACUM(I).EQ.COMMA)
     1GO TO 20
      IF(ACUM(I).NE.HYP) GO TO 50
   20 CONTINUE
   21 GO TO 60
   30 RETURN
   40 RETURN 1
   50 RETURN 2
   60 RETURN 3
      END


      SUBROUTINE TRAV
      IMPLICIT INTEGER*2 (A-Z)
      DIMENSION TOP(603),RIGHT(602),DOWN(601)
      DIMENSION LEVEL(50),NODTAB(50)
      DIMENSION SERCH(30),MINDEX(500,8),SEQ(4000),ACUM(241)
      COMMON/ONE/TOP,AVAIL1,P,Q,RR,DD,YY
      COMMON/THREE/ANS,SERCH,MINDEX,ACUM,SJ,SK,N,SEQ
      COMMON/FOUR/NODTAB,FD
      EQUIVALENCE (TOP(3),RIGHT(2),DOWN(1))
      DATA LX/'L'/,BLANK/' '/
C-----THIS SUBROUTINE TRAVERSES THE FORMAT TREE AND LOCATES
C-----THE END NODES OF EACH BRANCH. THE CLASS CORRESPONDING
C-----TO EACH END NODE IS STORED IN THE ARRAY 'NODTAB'.
      DO 10 I=1,50
   10 NODTAB(I)=BLANK
      I=0
      AVAIL1=MINDEX(SJ,8)
      RR=DOWN(AVAIL1)
      L=1
      LEVEL(L)=AVAIL1
   12 DD=RR
C-----EACH LEVEL IN THE TREE IS ASSIGNED A NUMBER WHICH IS
C-----STORED IN A STACK.AS THE TREE IS TRAVERSED
C-----THE STACK IS PUSHED DOWN OR POPPED UP ACCORDINGLY.
   15 IF(DOWN(DD).EQ.0) GO TO 20
      L=L+1
      LEVEL(L)=DD
      DD=DOWN(DD)
      GO TO 15
   20 I=I+1
      NODTAB(I)=TOP(DD)
      MINDEX(TOP(DD),5)=LX
      IF(RIGHT(DD).EQ.0) GO TO 25
      DD=RIGHT(DD)
      GO TO 15
   25 IF(LEVEL(L).EQ.RR) GO TO 30
      IF(LEVEL(L).EQ.AVAIL1) GO TO 35
      DD=LEVEL(L)
      IF(RIGHT(DD).NE.0) GO TO 27
      L=L-1
      GO TO 25
   27 DD=RIGHT(DD)
      L=L-1
      GO TO 15
   30 IF(RIGHT(RR).EQ.0) GO TO 35
      RR=RIGHT(RR)
      IF(DOWN(RR).NE.0) GO TO 12
      I=I+1
      NODTAB(I)=TOP(RR)
      MINDEX(TOP(RR),5)=LX
      GO TO 30
   35 RETURN
      END


      SUBROUTINE MENT
      IMPLICIT INTEGER*2 (A-Z)
      DIMENSION SERCH(30),MINDEX(500,8),SEQ(4000),ACUM(241)
      COMMON/THREE/ANS,SERCH,MINDEX,ACUM,SJ,SK,N,SEQ
      COMMON/FIVE/MULT,MK,EK
      DATA OP/'('/,CP/')'/,STAR/'*'/
C-----'MK'SCANS THE DATA ELEMENT FROM ITS FIRST 'OP'
C-----(COUNTING 'OP'S'UNTIL THE FIRST CHARACTER IS
C-----ENCOUNTERED.THEN 'EK'SCANS FROM 'MK' (COUNTING 'CP'S'
C-----UNTIL AN 'OP'IS ENCOUNTERED. IF THE PARENTHESES ARE
C-----UNBALANCED WHEN 'EK'STOPS THEN A MULTIPLE ENTRY EXISTS
      PCTR=0
      MULT=0
    5 IF(ACUM(MK).EQ.OP) GO TO 10
      MK=MK+1
      IF(ACUM(MK).EQ.STAR) GO TO 30
      GO TO 5
   10 PCTR=PCTR+1
      MK=MK+1
      IF(ACUM(MK).EQ.STAR) GO TO 30
      IF(ACUM(MK).EQ.OP) GO TO 10
   15 EK=MK
   20 EK=EK+1
      IF(ACUM(EK).EQ.CP) PCTR=PCTR-1
      IF(ACUM(EK).EQ.STAR) GO TO 30
      IF(ACUM(EK).EQ.OP) GO TO 25
      GO TO 20
   25 IF(PCTR.NE.0) MULT=1
   30 RETURN
      END


      SUBROUTINE QSCAN(*,*,*)
      IMPLICIT INTEGER*2 (A-Z)
      DIMENSION UPRAN(30)
      DIMENSION SERCH(30),MINDEX(500,8),SEQ(4000),ACUM(241)
      COMMON/THREE/ANS,SERCH,MINDEX,ACUM,SJ,SK,N,SEQ
      COMMON/SIX/S2,AT,AE,HCTR,UPRAN
      DATA OP/'('/,CP/')'/,COMMA/','/,BLANK/' '/,HYP/'-'/,ST
     1   COL/':'/,AMP/'&'/
C-----THIS SUBROUTINE LOCATES KEYWORDS IN THE QUERY,
C-----DETERMINES IF ALPHABETIC/NUMERIC RANGES ARE REQUESTED,
C-----LOADS THE 'SERCH'ARRAY,AND RETURNS TO THE M/PROG FOR
C-----QUERY PROCESSING.
      S1=0
      S2=0
      AT=N+1
   10 IF(ACUM(AT).NE.OP) GO TO 15
      AT=AT+1
      GO TO 10
   15 HCTR=0
      IF(ACUM(AT).EQ.HYP) GO TO 40
   20 AE=AT
      IF(S2.EQ.1) GO TO 22
      IF(S1.EQ.0) HCTR=HCTR+1
   22 DO 25 I=1,30
      SERCH(I)=BLANK
      UPRAN(I)=BLANK
   25 CONTINUE
      N=1
   30 SERCH(N)=ACUM(AE)
      AE=AE+1
      IF(ACUM(AE).NE.COMMA.AND.ACUM(AE).NE.CP) GO TO 32
      S1=0
      S2=0
      GO TO 50
   32 IF(ACUM(AE).NE.COL) GO TO 34
C-----ARRAY 'SERCH' IS LOADED WITH THE LOWER RANGE LIMIT
C-----AND ARRAY 'UPRAN IS LOADED WITH THE UPPER RANGE LIMIT.
      S1=1
      UI=1
      AE=AE+1
   31 UPRAN(UI)=ACUM(AE)
      AE=AE+1
      IF(ACUM(AE).EQ.COMMA.OR.ACUM(AE).EQ.CP) GO TO 60
      UI=UI+1
      GO TO 31
   34 IF(ACUM(AE).EQ.AMP) GO TO 35
      N=N+1
      GO TO 30
   35 S2=1
      GO TO 50
   40 HCTR=HCTR+1
      ENTRY QSCAN1(*,*,*)
   45 AT=AT+1
      IF(ACUM(AT).EQ.STAR) GO TO 70
      IF(ACUM(AT).EQ.OP.OR.ACUM(AT).EQ.COMMA.OR.ACUM(AT).EQ.
     1   GO TO 45
      IF(ACUM(AT).EQ.HYP) GO TO 40
      GO TO 20
   50 RETURN 1
   60 RETURN 2
   70 RETURN 3
      END
```

BIBLIOGRAPHY

1. Taube, M., "Progress in the Design of Information Retrieval Systems," Advances in EDP and Information Systems, American Management Association Report No. 62, 1961.

2. Bourne, C. P., Methods of Information Handling, Wiley, 1963.

3. Kildall, G. A., Experiments in Large Scale Computer Direct Access Storage Manipulation, Computer Science Group, University of Washington, Seattle, Technical Report No. 69-1-01, January 8, 1969.

4. Meadow, C. T., The Analysis of Information Systems: An Introduction to Information Retrieval, Wiley, 1967.

5. Hays, D. G., "Research Procedures in Machine Translation", Natural Language and the Computer, p. 183-214, McGraw-Hill, 1963.

6. Harper, K. E., "Dictionary Problems in Machine Translation," Natural Language and the Computer, p. 215-222, McGraw-Hill, 1963.

7. Garvin, P. L., "Syntax in Machine Translation," Natural Language and the Computer, p. 223-232, McGraw-Hill, 1963.

8. Mersel, J., "Programming Aspects of Machine Translation," Natural Language and the Computer, p. 233-251, McGraw-Hill, 1963.

9. Salton, G., "A Comparison Between Manual and Automatic Indexing Methods," American Documentation, v. 20, no. 1, p. 61-71, January, 1969.

10. Travis, L. E., "Analytic Information Retrieval," Natural Language and the Computer, p. 310-353, McGraw-Hill, 1963.

11. Green, B. T., and others, "Baseball: An Automatic Question Answerer," Computers and Thought, p. 207-216, McGraw-Hill, 1963.

12. Weissman, C., LISP 1.5 Primer, p. 5-22, Dickenson, 1968.

13. Computer Facility, Naval Postgraduate School, CP/CMS User's Guide, September, 1969.

14. Becker, J. and Hayes, J. M., Information Storage and Retrieval: Tools, Elements, Theories, Wiley, 1963.

15. Williams, W. F., Principles of Automated Information Retrieval, The Business Press, 1965.

16. Artandi, S., An Introduction to Computers in Information Science, The Scarecrow Press, 1968.

17. Janker, F., Indexing Theory, Indexing Methods, and Search Devices, The Scarecrow Press, 1964.

18. Swets, J. A., "Effectiveness of Information Retrieval Methods," American Documentation, v. 20, no. 1, p. 72-89, January, 1969.

19. Kellogg, C., "The Fact Compiler: A System for the Extraction, Storage, and Retrieval of Information," Proceedings of the Western Joint Computer Conference, vol. 17, p. 73-82, 1960.

20. Martin, J., Design of Real-Time Computer Systems, Prentice-Hall, 1967.

21. Withington, D. G., The Use of Computers in Business Organizations, Addison-Mesley, 1966.

UNCLASSIFIED
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) Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE

Application of a Data Structuring Concept in a General-Purpose Fact-Retrieval System

4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)
Master's Thesis; September 1970

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

Richard Joseph Petrucci

6. REPORT DATE September 1970	7a. TOTAL NO. OF PAGES 94	7b. NO. OF REFS 21
8a. CONTRACT OR GRANT NO. b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

An on-line, general-purpose fact-retrieval system is presented which employs a classificatory data structuring technique. The technique embraces the basic concept of hierarchical classification of data and provides users with multiple anenues of access to a data file. Additionally, the data file may be partitioned into unrelated data sets.

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6811

UNCLASSIFIED
Security Classification

A-31408

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Information retrieval						
Hierarchical classification						
Indexing						
Fact retrieval						
Computer						
Storage						