THE DESIGN OF A BIBLIOGRAPHIC DATA BASE SYSTEM

By

PERRY LEE BALL

Bachelor of Science

Northwestern State University of Louisiana

Natchitoches, Louisiana

1973

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
MASTER OF SCIENCE
May, 1975

THE DESIGN OF A BIBLIOGRAPHIC

DATA BASE SYSTEM

Thesis Approved:

G. E. Hedrick
Thesis Adviser

Donald W. Grace

James R. Van Doren

D. D. Durham
Dean of the Graduate College

PREFACE

This report presents a definition of a data base and describes the major concepts considered in the design of a data base system. Demonstrating these concepts is an implemented data base system involving bibliographic data pertaining to computer science topics.

I wish to thank Dr. J. R. Van Doren and Dr. D. W. Grace for their suggestions for improvement of this report. A special thanks goes to my adviser, Dr. G. E. Hedrick, for his valuable guidance and assistance.

My final thanks goes to my wife, whose patience, understanding, and encouragement played an important role in my completion of this last year of study.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Modern society continually produces and uses information. Most technical activity--in science, industry, commerce or government--now takes place in such a complex environment that it must be based on specially acquired information. At the same time, every act gives rise to information and recorded knowledge steadily grows. Finding the information one seeks within the huge mass now available becomes more difficult. If information is to be readily accessible it must be organized.

Traditionally, data files have been assembled to serve individual applications, such as inventory control, payroll, manufacturing planning, and so on, each data file having been specifically designed with its own storage space within the computer, on tape, or direct access device. In many instances these data files included duplicate or redundant information. This often resulted in one file being current while the other remained static. Since each application program was tailored for its particular data file, extensive revision for each program was necessary whenever new types of information were introduced for the data set or new data management techniques become available; therefore, application programs could be in an almost perpetual state of change, adding appreciably to the overall cost of data processing.

These undesirable attributes of constant revision and higher data processing cost for data files have been eliminated to some extent by the advent of the "data base." The term data base does not have a single accepted definition and is quite often defined as whatever the designer chooses to call a data base. This report will adhere to the definition of a data base as a set of one or more files containing "nonredundant" interrelated data items which are processable by one or more applications (5, 9, 15, 16, 20). The goals achieved by a good data base system are:

. Elimination of redundant data and implied consequent maintenance

. Consistency through the use of the same data by all applications

. Addition of data to an existing data base without modifications of existing application programs

. Reduction of data processing cost.

By ineluding with the data base facilities for file structuring, file updating, file modification, file interrogation, and report generation, a data base system is developed. The system described thus far is normally termed a generalized data base management system (GDBMS or GDMS). A variety of such systems are in operation today--for example, GIS (3, 5, 16), IDS (3, 5, 16), IMS (5), TDMS (2, 3, 5, 16) and many others. Various other generic names used in the literature for GDMS are data management systems, information management systems, and file management systems. If the system is not generalized it may be identified as "tailored." This implies that the data base is limited to data from a single subject and built for a specific application. For instance, an insurance package may be tailored to the processing of data concerning insurance policies, such as account number, renewal data, policy type, coverage, and so on. Another example is a "bibliographic" data base system which might be used by a library to process the information on the documents it contains, such as call number, author's name, abstracts, and so on.

The primary objective of this paper is to present the concepts of a data base management system. A second object is to demonstrate some of these concepts in a practical application involving a bibliographic data base system.

In 1969 a study concerning GDMS was undertaken by the Codasyl System Committee. From this investigation have come two reports (4, 5). The first report described a list of the features that belong to GDMS; the second report expanded this list and described the features in further detail as they are supported in implemented systems.[1] The later report established a distinct difference in the manner in which a GDMS provides facilities for the user to manipulate data. Because of this distinction GDMS's are separated into two classes. It is this classification that is presented in Chapter II. The major features that comprise a data base system

---

[1]Minker (16) also gives a review of several systems and their features.

are examined in Chapter III and Chapter IV, with the final data base concepts--the data base functions--being investigated in Chapter V. Since the above concepts comprise a complete overview of a data base system, the design of a particular data base system for a specific application may be evaluated.

As in many bibliographic data base systems, such as NASA/RECON (27), TIP (MIT) (27), and SKI-KWOC (Ames) (10,27), subject analysis of the documents in the data base is required. Therefore, Chapter VI explains some of the techniques available in performing the analysis.

In pursuing the second objective of demonstrating the major concepts of a data base system with a practical application, Chapter VII contains the description of an implemented bibliographic data base system. This is followed by the final chapter which provides a summary of the data base concepts presented in this paper and of the implemented bibliographic data base system. Also included in this chapter are recommendations for improvements to the implemented data base system.

The Appendices are provided to assist the reader, the user of the data base system, and the data administrator for the system. Definition of terms used in this paper pertaining to data bases and to the implemented bibliographic data base system are provided for the reader in Appendix A and Appendix B, respectively.

In Appendix C, The User's Guide, requirements for interrogating the bibliographic data base are described. Appendix D examines in detail the features and functions of the bibliographic data base system. Therefore, it is used as the Data Administrator's Guide. Following the Administrator's Guide is Appendix E which provides flowcharts for all algorithms of the data base system and Appendix F which contains sample output from the implemented data base functions.

CHAPTER II

CLASSIFICATION OF DATA BASE SYSTEMS

In the second report of the Codasyl System Committee a significant distinction exists between system capabilities which are provided through system languages tailored to particular functions and capabilities which are provided by augmenting a general purpose language, such as Cobol or PL/1 (5). Capabilities in the former category are labeled self-contained capabilities and those in the latter category are called host language capabilities.

Data base systems are classified into two groups according to the type of capabilities they support. These are self-contained systems and host language systems. Unless explicity stated otherwise, both types of systems incorporate the features described in Chapter III and Chapter IV in some form.

Host Language Systems

A host language system may be regarded as a conventional procedural language, which provides the added capability of manipulating a data base. The host language features comprising the added capabilities are described as programming facilities of a data base system (These are discussed in detail in Chapter III). These features do not constitute a complete language and must co-exist with the procedural language (referred to as the host language). Programming facilities which facilitate the use of more complex data structures are not easily obtained with the host language alone and for control and manipulation of data not stored in primary storage. An individual using a host language system is still considered to be an application programmer, because he specifies the logical flow of the system through a coded set of statements. Although the programmer is insulated from the physical storage structures, he has the added flexibility of manipulating the data base with his own procedures. The degree of flexibility allowed a user in manipulating a data base varies from one system to another. Usually, this flexibility corresponds directly with the degree of responsibility inherited by the user for maintaining the integrity of the data base.

Self-Contained Systems

In self-contained systems, sometimes classified as non-procedural, the user does not exercise control over the sequence of detailed steps the system uses to process the requirements. The basic aim of such a system is to make the system easier to use for an individual who does not have conventional programming expertise. The minimization of time and writing required by a user is achieved through special functions that are pre-programmed (or built-in processing algorithms) and may be treated as primitive functional capabilities. To invoke these functions each self-contained system must supply a set of commands often referred to as the query language or the user language. These commands reduce the duties of the non-programmer to invoking predefined algorithms and possibly providing values to any parameters they may require. A major disadvantage of easy-to-use systems is their lack of flexibility. The user must use only system routines to perform all of his functions, thus limiting the set of applications which can be handled.

CHAPTER III

DATA BASE FEATURES I

Whatever name a generalized data base management system carries it should provide the following data base features: 1) a description of the data, 2) logical structures for representing the data, 3) physical representation of the data on a storage medium and means of accessing it, and 4) the assigned responsibility of data base management (3, 5, 16). In the early days of computing these features were subordinate to program development. In fact, the focus for a particular application tended to be on the design and development of a computer program to perform the required function, the organization of the data being a secondary consideration. The data files in a data base system should be organized in a fashion that permits their use in several applications rather than a single application, thus new programs are designed using the previously established data base. No matter how these features are implemented and emphasized among themselves, the principal objective of a data base system is to provide an efficient procedure in terms of time and storage for the manipulation of data in a variety of situations.

Of the four features described for a data base, normally the one most often encountered by the user is the logical structure used to represent the data information. These structures are referred to as data structures. Because of their importance in determining the usefulness of the data base to the user, data structures are considered separately in this chapter. Although the remaining features are treated in the following chapter, they are not considered any less important to the data base system.

Data Structures

Data structures in a data base are determined by the logical arrangement of the information it contains and the means provided to enable the accessing of logically related records. The logical structure of a data base is one of the most important factors in determining the usefulness of the data base to the user.

A user can establish a "logical" data base consisting of subsets and concatenation of parts from the physical data base. When the storage structure of the physical data base is defined it must have provisions for all of the information needed to implement the logical data base. Most of these provisions take the form of pointer fields which set up logical connections between items within the same or different physical data base. The results of this linkage is a complex network of a physical data base forming a system data base (This is illustrated in the implemented system). Data structures which facilitate the manipulation of logical relations by pointer fields are known as linked structures.

A careful study of data structures used within a data base system to assist the user will reveal that some type of linked structure is usually incorporated. The most common linked structures are singly linked list, doubly linked list, circular list, multilinked list, inverted list, and tree structures. Structures other than those mentioned are possible for data bases, but upon investigation they frequently are found to be either combinations of methods examined in this section, or different names for the methods described here.

It should be noted that a data base system is not limited to any one type of linked structure. In fact, it is more likely for a particular system to employ different combinations of linked structures mentioned. Although the criterion for selecting a specific linked structure may be rapid access to data, the traditional space/time trade-offs that are common for all computational techniques and structures cannot be ignored.

Linked Structures

With this type of data structure related data items need not be stored in consecutive memory locations, but can be scattered throughout the storage medium. In essence a linked list structure is composed of numerous items called "nodes," each containing one or more "fields." A field can contain a data item, such as a character string, fixed or floating point value, or it can contain a pointer to another node. A pointer or link is merely a means of identifying and manipulating a particular node. Representation of a node on the storage medium usually consists of a single "block" of storage in which its fields are stored side by side. The physical address of this block of storage is then utilized as a pointer.

The various nodes in a linked list structure need not be the same size. In fact, the nodes may even contain different types of fields. However, it is considerably more difficult to handle a list which has a varying node structure. Storage allocation and processing are much easier for a linked list if all its nodes are the same size and contain the same types of fields. Many systems, including the one reported in this paper, limit their linked structures to those which have a uniform node structure.

Efficient utilization of memory space is one of the more significant advantages of linked lists. Whenever a linked structure is used a special list of unused nodes called the availability list can be maintained (7, 11, 13). By removing nodes from this list and linking them as required, new lists are formed and new nodes can be added to existing lists. On the other hand, when a particular list, or part of a list, is no longer needed, it can be returned to the availability list. Subsequently, the same area may be reused to construct a new list or new nodes for the old list. Close control over the utilization of nodes allow free space to be returned to the availability list as soon as the nodes are no longer needed resulting in a significant saving in the amount of storage used. Such a cleanup operation is often called "garbage collection" (11, 13).

Singly Linked List

The type of linked list structure, where each node contains one pointer to indicate the next node in the list, is called a singly linked list (sometimes referred to as "chaining") (7, 11, 17, 20). The use of pointer with each node to identify the next node in the list allows the various nodes to be scattered in storage rather than in precisely the same physical order as that in which they are processed. Identification of a unique list is accomplished by assigning a single variable representing a pointer to the location of the first entry of the list.

Figure 1 is an example of a singly linked list in which the records are stored in a logically sequential order. They are not necessarily in the same physical order. The first record is at location 10, the second record at location 30, and the third at location 45. The logically sequential organization is obtained by using the links shown in the diagram. "Head" is the variable containing the first entry of the list. In the last record (Record 3) of the list the pointer value is $\perp$. This special pointer value is called the "null" symbol. The null symbol is a value that cannot be confused with any valid pointer value. Since the null value cannot be interpreted as pointing to another node, we can employ such a device to identify the last node in the list.

Doubly Linked List

The number of links in a record is not limited to one; a varying number of links may be included in each record for different list structures. One such structure is a doubly linked list illustrated in Figure 2. A doubly linked list contains two link fields per node, the first field pointing to the successor node and the second pointing to the predecessor node. It is evident that the doubly linked list does not represent any more structural information than its equivalent singly linked list; however, the use of extra space required to store the links for a doubly linked list is justified by two advantages: the list can be searched in either a forward or backward direction, and insertion and deletion of nodes to the right or left of any node can be accomplished very easily. For example, if we use only a single link, the recognition of the predecessor of a particular node requires

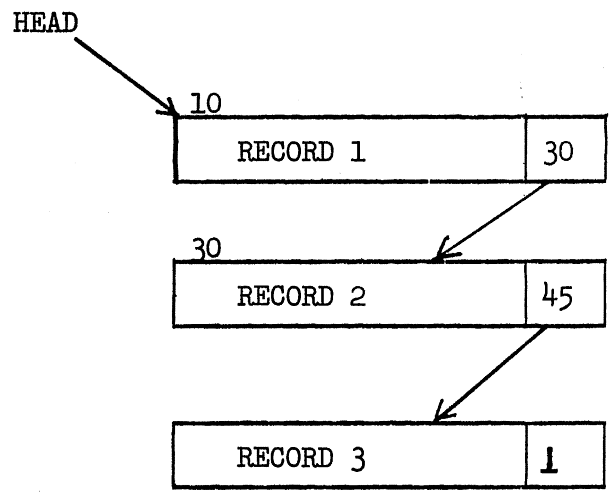


Figure 1. Singly Linked List

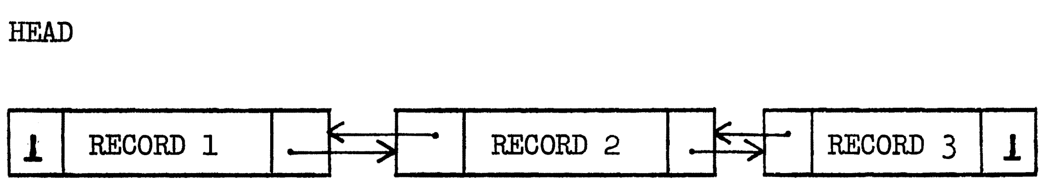


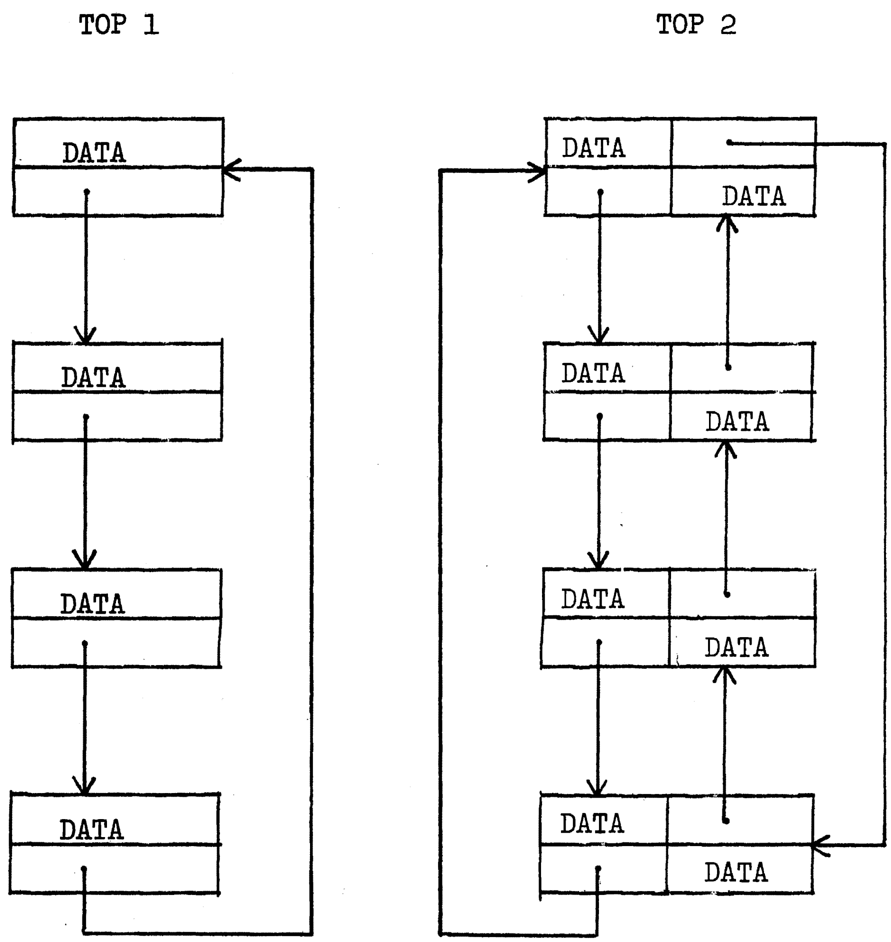
Figure 2. Doubly Linked List

a linear search, beginning at the top node and progressing down through the list until a node is encountered whose link points to the designated node. If the list is a long one this can be a very costly and time-consuming operation. The inclusion of a backward pointing link eliminates this linear search at a cost of additional storage of a second link in each node.

Circular Linked List (Rings)

A singly linked list may be used in forming circular linked list--sometimes referred to as rings. A circular list is simply a singly linked list that closes upon itself (7, 11, 17, 20). This is accomplished by replacing the null pointer of the last node's link field with the location of the head of the list. Figure 3 displays a singly linked list and doubly linked list in circular form.

The advantage of such a structure stems from the fact that each node is accessible from any other node in the list. This characteristic makes the ring structure especially suitable for applications in which each record in a series must be processed. An operational data base system has been designed with these features (2, 5, 16). The system is known as Integrated Data Store (IDS), produced by General Electric (GE). IDS is a set of subroutines for processing rings; facilities to create and destroy, add to and delete from rings are provided (2, 5, 16).



a) Singly Linked List

b) Doubly Linked List

Figure 3. Circular Linked List

Inverted List

An inverted list structure maintains an ordered list of the key terms extracted from search elements. Each value of the key term contains an address of link to the particular data base entries that contain the specific key (See Figure 4) (3, 7, 13, 17, 20). If an inverted list is kept on all fields of a record, the record is said to be "completely" inverted. When only selected fields are inverted, the term partially inverted is commonly used. The benefit of a completely inverted structure is the ability to access the system through any desired field while partial structures only allow selected key fields. A data management system that has been designed around the inverted list structure is TDMS (Time-Shared Data Management System) produced by the System Development Corporation (5, 16, 24). TDMS uses completely inverted files with a hierarchical dictionary of several levels that is used to locate the inverted lists (5, 16, 24).

Since an inverted list contains all references to a particular key in the data base system, a major advantage of rapid access only to the desired records is achieved. A second advantage, that of union and intersection operations on inverted lists, is facilitated if each list is maintained in collating order of the record addresses. In this case two lists can be intersected or their union can be found with one pass through both lists. To illustrate this facility, assume that a query consisting of the conjunction of two items X and Y is required. The list of addresses for item X is intersected with the list of addresses for term B. This results in precisely the set of addresses that must be accessed in the file to satisfy the XY query.

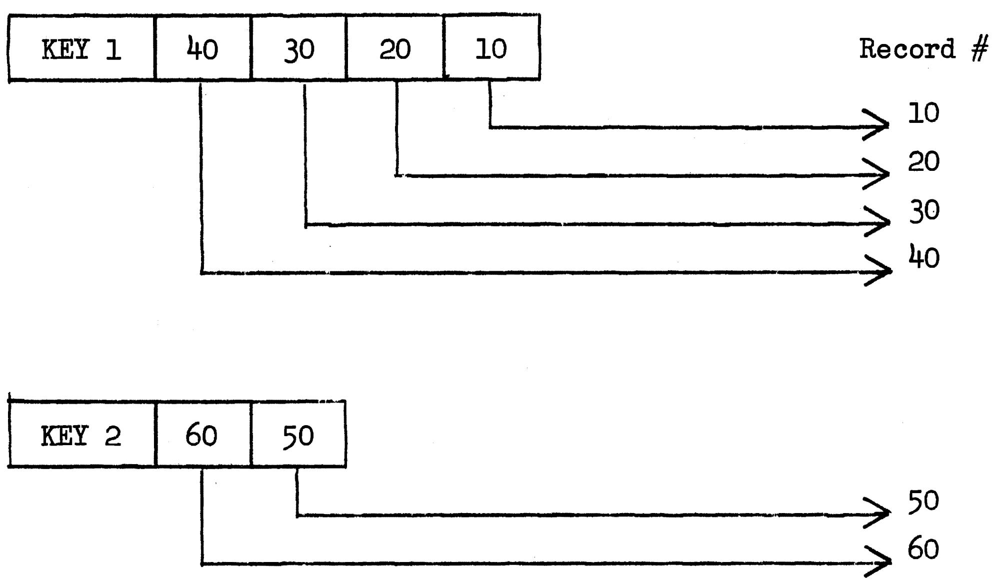


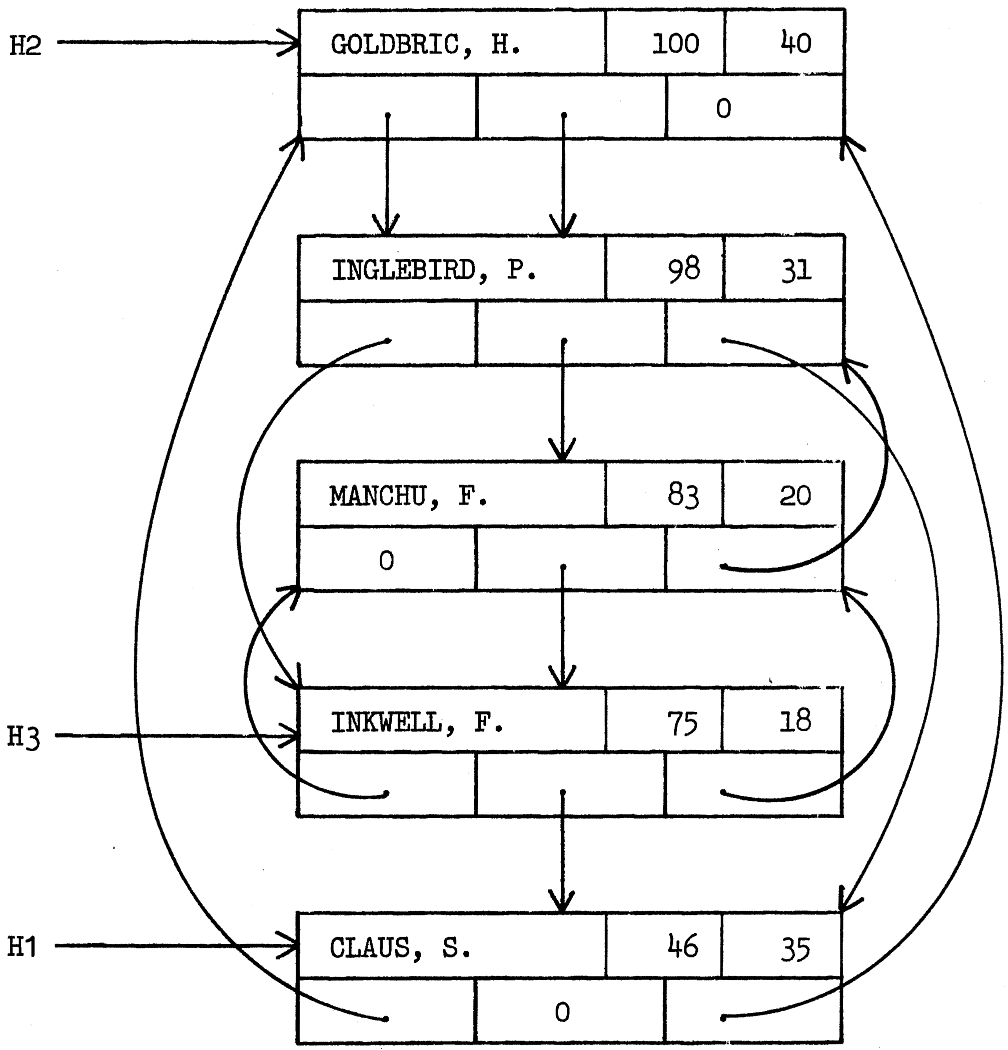
Figure 4. Inverted List

The major disadvantage of inverted list structures is that they are difficult to update. There are two factors contributing to this problem. First, blocks of reserve space must be maintained in the key term index to allow for the address list expansion or alternatively, a complicated scheme of linking blocks must be programmed. The second factor is that new addresses must be inserted in sequence in order to retain the efficiency of the union and intersection process. Therefore, inverted list structures are ideal for applications that require rapid retrieval of information and a relatively low update volume.

Multilinked List

Multilinked list or knotted list consists of a sequential index that gives for each key the location of the start of a list that links together all records characterized by that key value (7, 11, 13, 17). A multilinked structure is conceptually an inverted file, where only the heading of each inverted list is kept in the index and the rest of each list is indicated by links among the records.

As an illustration, suppose we consider a class of students for which we would like to have three different lists -- one classified according to student surnames sorted in alphabetical order, a second arranged in descending order according to final marks, and a third listing the student in ascending order of age. Each student in the class can be represented by a single record which contains six fields, one each for his name, final mark, and age, and three links. Each serves as a link for a different order. Three pointers must be maintained--one to the node representing the student whose name appears first in the alphabet, H1; a second to the node representing the student with the highest score, H2; and a third identifying the node associated with the youngest student, H3 (See Figure 5). To proceed through the list in alphabetical sequence, one begins with the node designated by H1, and follows the pointers in link 1 until we eventually reach the node containing the name which appears last in the alphabet (i.e., the node in which link 1 has the null symbol 0). Similarly for traversing the list in descending order of marks or ascending order of age, we begin with node H2 or H3, and follow link 2 or link 3 respectively.



NAME		MARK	AGE
LINK 1	LINK 2	LINK 3	

Figure 5. Multilinked List

Although multilinked structures were compared, conceptually, to inverted list structures the particular file characteristics of retrieval and updating are reversed.[1] Multilinked structures are easier to update than the inverted list because they avoid the necessity for complete reorganization of the sequentially allocated inverted list; however, retrievals are slower for the multilinked structures because the lists must be traversed to perform a retrieval.

Tree Structures

Tree structures comprise an important class of data structures included in information retrieval techniques. Trees sometimes are used as indexes to some other file. Such an arrangement permits a record in the tree to consist of only keys, pointers to other records in the tree, and addresses in the file. This approach is particularly useful if the records in the file are variable in length. If the file consists of short, fixed-length records, then the entire record can be placed within the tree structure.

Knuth (4) states that:

> A tree is a finite set of one or more nodes such that,
>
> (1) There is one designated node called the root of the tree;
>
> (2) A set of directed arcs leading from the root node to m other nodes;
>
> (3) The remaining nodes are partitioned into $m \geq 0$ disjoint sets $T_1, \ldots, T_m$ corresponding to

[1]Lefkovitz (13) gives a detailed comparison of multilinked list and inverted list characteristics.

> each of the nodes specified in (2), and each in turn is a tree. The trees $T_1, \ldots, T_m$ are termed subtrees of the root (p. 305).

From the definition it is evident that every node of a tree is the root node of some subtree contained in the entire tree. The number of subtrees of a node is called the degree of the node. A node of zero degree is called a terminal or leaf node. The level of a node is defined by saying that the root has level 0, and other nodes have a level that is one higher than they have with respect to the subtree T_J, of the root, which contains them. The maximum number of levels in a tree is defined as the height of the tree.[2]

The above ideas are illustrated in Figure 6 which depicts a tree with ten nodes. The root is A, and it has four subtrees {BF}, {CGH}, {DI}, and {E}. Node C is on level 1 with respect to the root A and has two subtrees G and H ; thus C has degree 2. The terminal nodes of the tree are F, J, H, I, and E; all of which have degree 0. The height of the tree with root A is 4.

---

[2]The terminology established here is in accordance with Knuth (11).

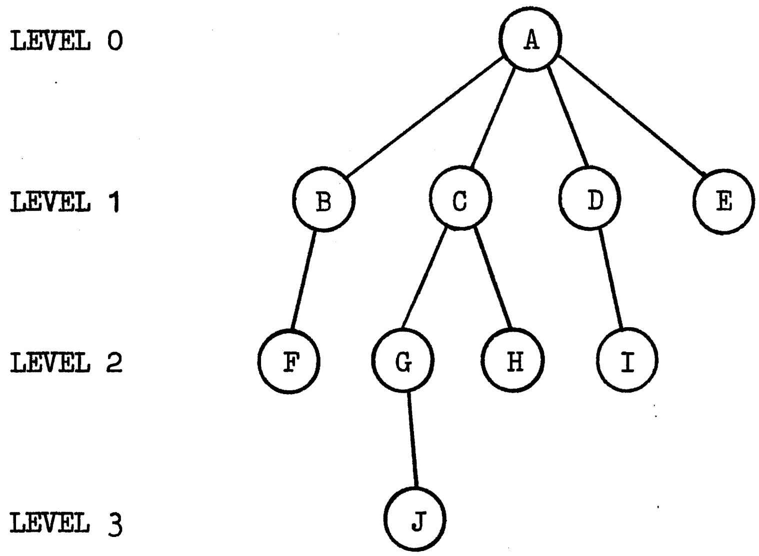


Figure 6. A Tree

Binary Trees

A binary search may be used for a sequentially allocated random-access file that is stored in order of the collating sequence of its keys. This arrangement reduces search time at the expense of update time. For a file that is updated more often than it is searched, linked allocation can be used to minimize update time at the expense of search time. For a file that is updated and searched with similar frequency, however, neither of these approaches is very practical, and some compromise may be desirable. A binary tree structure is such a compromise; it combines the speed of a binary search, an average of $\lfloor \log_2 N \rfloor + 1$ (where N is defined as the number of items in the structure and the symbol $\lfloor X \rfloor$ indicates the largest integer that does not exceed X), with the updating ease of linked allocation (11, 25).

Knuth (11) defines a binary tree "as a finite set of nodes which is either empty or consists of a root and two disjoint binary trees called left and right subtrees " (p. 315). Since a binary tree is either empty or contains a root node with two binary subtrees, each node within the tree will contain a key value and at least two pointers, referred to as a left link and a right link. With some binary trees there exist a predefined relationship in terms of collating sequence among the keys in the nodes.[3] In such a binary tree the left link points to a subtree which contains key values that are less than the key value in the root node and the right link points to a subtree which contains key values that are greater than the key value in the root node. If such a subtree does not exist, the corresponding link field will contain the designated null symbol.[4] Figure 7 demonstrates such a relationship among the keys in a node.

If a tree is allowed to change without restriction it may degenerate into a singly linked list. When this occurs the average search time for retrieval can increase from the desirable order of $\log_2 N$

---

[3]Knuth (11) refers to this type of tree as a binary search tree.

[4]Knuth (12) describes cases where the null symbol is replaced by a pointer to some other part of the tree. This is known as a threaded tree. The present discussion does not include such trees.

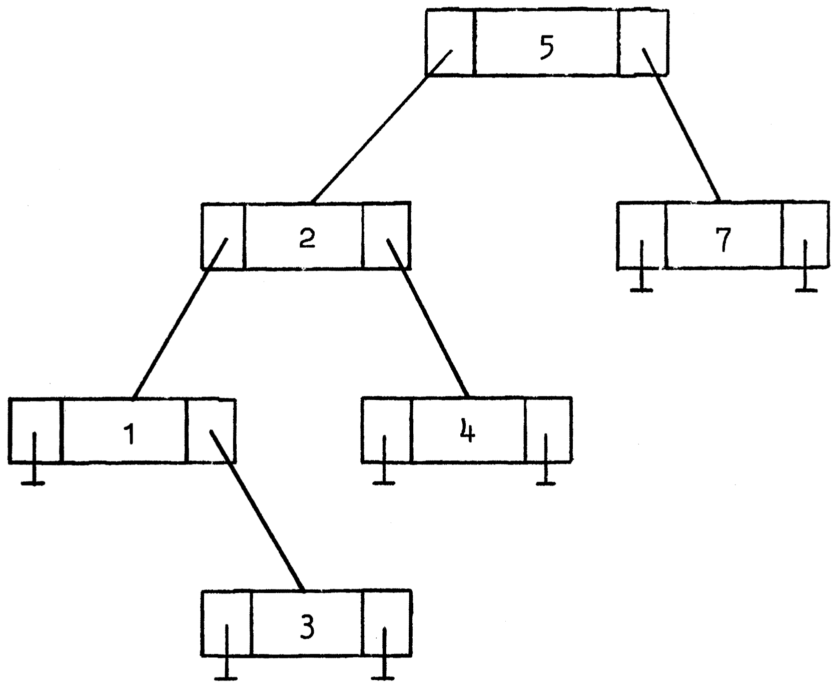


Figure 7. A Binary Tree

to the undesirable average of N/2. The term used to describe such a fluctuating binary tree is "unconstrained" or "unbalanced." Figure 7 shows an example.

In order to effect a compromise between unconstrained trees and optimum binary trees--trees which contain all leaf nodes on at most two adjacent levels--it may be stipulated that the left subtree of a node may not differ in height by more than some number, called the balance factor, from the height of the right subtree. With this restriction the minimun average search time is retained and a "constrained" binary tree results. G. M. Adel'son-Vel'skit and

E. M. Landis proposed that the balance factor be held at one. From this proposal they developed a constrained binary search tree known as the AVL tree.[5]

AVL Trees

AVL trees may be defined formally as a balanced binary tree in which the length of the left subtree differs by at most one from the length of the right subtree (12, 22). The price paid for a guaranteed upper bound on searches provided by AVL trees is the extra storage needed for a balance tag in each node (See Figure 8) and programming complexity for restructuring the tree after insertions and deletions.[6] This guaranteed upper bound of the maximum number of probes, when searching for a particular key, can be established to be about $1.5 \log_2 N$ (12, 22). Although empirical evidence shows that the expected average for a search is around $(\log_2 N) - 0.75$ and for insertion the average is one greater than retrieval or $(\log_2 N) + 0.25$ (22), J. R. VanDoren and J. L. Gray (22) have shown empirically that AVL trees have desirable updating features. In fact, the average number of transformations required to maintain the AVL balance after insertion is approximately 0.5 and an average of 0.23 for deletion (22). This seems to indicate that the maintenance of an AVL tree which is subject to frequent modifications by virtue of insertions and deletions is

---

[5]A description of other types of constrained binary search trees can be obtained by consulting Knuth (12).

[6]A balance tag is an indicator used in maintaining the AVL balance conditions.

within an acceptable range of minimizing the updating versus searching conflict. This establishes the AVL tree as a good choice for indices to data recorded on direct access devices. Although AVL trees may be kept on external storage, they are primarily designed to be maintained in internal memory.

LLINK	RLINK	BALANCE TAG	INFORMATION

Figure 8. Typical Node for an AVL Tree

If the restriction that trees allow at most two branches from the root node is abandoned, allowing the root node to have any number of branches extending from it, a multiway tree is developed. The effect of such a change results in a decrease in the number of levels in the tree (and consequently, the number of probes) at the expense of increasing the number of branches from each node. A multiway tree that exhibits these features is the B-tree (4, 6).

B-Trees

A B-tree is an alternate way of organizing information on an external device which facilitates rapid access and maintenance. Each particular B-tree has a predetermined maximum number of branches from each node. This number in turn determines the order of the tree. Other unique qualities for B-trees of order m are:

1) Every node has at most m sons.

2) Every node, except the root and the leaves has at least $\lceil m/2 \rceil$ sons (the symbol $\lceil X \rceil$ indicates the smallest integer that is greater than or equal to X).

3) The root node has at least two sons, unless it is a leaf, in which case it is the only node in the tree.

4) All leaves will have null pointers and will be on the same level, which in fact will be the bottom level of the tree.

5) A non-leaf node with K sons has K - 1 keys. This property along with the first two implies that every node, except the root, will contain between $\lceil m/2 \rceil$ - 1 and m - 1 keys. (6)

Figure 9 shows a B-tree of order 4. Each node has more than m/2 sons, and the root node which may contain from 1 to 2 keys, has 2 keys. Also, all leaves are on the same level (2) and contain the null pointers.

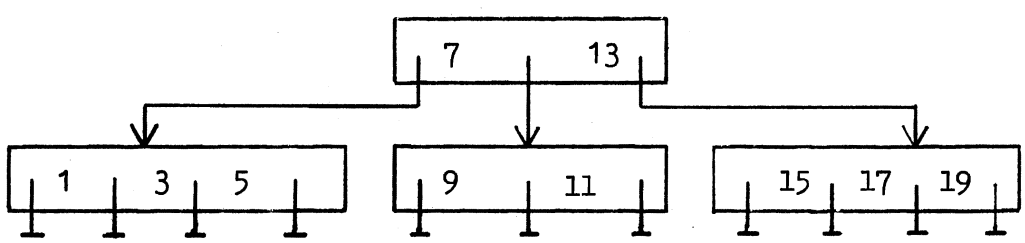


Figure 9. Order 4 B-Tree

As with AVL trees a guaranteed efficiency for updating and searching large files exists. With a tree of N keys the upper bound (U) on the levels of the tree may be evaluated by using the following formula (4,6).

$$U \leq 1 + \log_{\lceil m/2 \rceil} \left(\frac{N+1}{2}\right); \quad \text{where,}$$

U - upper bound on levels of the tree;

m - order of the tree;

N - number of keys in the tree.

The maximum number of probes (also defined by U) depends on the number of keys and the order of the B-tree. Therefore, there is a trade-off between the number of levels in the tree and the size of the nodes. Results of this trade-off must be weighed very heavily for each application because they affect: (1) the node occupancy ratio of the number of keys in a node to the maximum number of keys possible per node; (2) the reorganization required within a node; and (3) the organization required among nodes (6). A choice of the order of a B-tree and construction of nodes must be investigated thoroughly for a particular application.

Summary

Data structures may be considered to be the users concept of the data in a data base. They allow the user of a data base system to interact with the data in a manner that is independent of the physical storage of the data. Frequently data structures use pointers, which are merely addresses to retain the logical relationship between items. Logical structures of this nature are called linked structures. The more common linked structures are singly linked list, doubly linked list, circular list, multilinked list, inverted list, and tree structures. Since each of these structures possesses advantages and disadvantages for searching, updating, and storage of information, it is necessary when designing a data base system to investigate each structure (or combination of structures) to determine the optimum solution for a particular application.

CHAPTER IV

DATA BASE FEATURES II

The data base features that may be less apparent to the user are: 1) a description of the data, 2) the physical representation of the data on a storage medium and means of accessing it, and 3) the assigned responsibility of data base management. In keeping with the terminology provided by the Codasyl Systems Committee these features are referred to as data definition, storage structures, and data administrator, respectively (4, 5).

Data Definition

The data definition process for many self-contained systems is restricted to nothing more than the actual description of the items in a record for a particular application. Figure 10 will serve as a representation of such a strict definition. In this illustration "record" is being defined as containing a "name" field that is fifty characters long, and a five digit number field called "zip-code."

To obtain a more complete overview of data definition within host language systems it will be described as a process consisting of a tightly knit relationship between a data description language (DDL), a data manipulation language (DML), and the data base management system (DBMS) (4, 5).

01 Record		
02	Name	Character (20)
02	Address	Character (50)
02	Zip Code	Picture 99999

Figure 10. Data Definition

The DDL is the language used to declare the schema. A schema, in simple terms, may be considered to be a master catalog which indicates the type of organization in each file, the information contained with each file, and the inter-record and inter-file relationships in the data base. A subset of a schema, which names only the information for one or more application programs, is known as a subschema. Schemas and subschemas once written in a DDL have the capability of being compiled separately from any user program and stored in a library. Therefore, DDL provides for data independence between the application program and the data description.

The DML is the language which the programmer uses to cause data to be transferred between his program and the data base. Although DML's initiate data transfers, all interfaces of the DML with data in the data base are at the symbolic or logical level. These languages are nothing more than extensions of a host language and not complete languages in themselves. DML's rely on a host language to provide the framework for them and to provide the procedural capabilities required to manipulate data in primary storage.

Data base management systems consist of interface programs which provide for the access of the data base. It is here that the schema and subschema are analyzed and combined to form the required data transfers. Thus, all physical transfers of data in the data base are handled by the DBMS while the DML operates on the logical level with the data base.

The relationship between the DDL, DML, and DBMS may be best explained by tracing the call for data by a user program. All calls to the DBMS originating from the user programs are made in the DML. It is then the function of the interface programs to analyze the call on the basis of the schema and subschema, and request physical I/O operations from the operating system. The data is obtained by the operating system and placed into the system's buffers. DBMS interface programs then perform the final transfer (and possibly data conversions) of data between the system buffers and the user program (4, 5, 20). It should be understood that these functions described are those of host language systems and not of most self-contained systems. Briefly summarized, these functions consist of: a DDL which describes the data referenced by a program, a DML which is the program's interface with the data base, and the DBMS interface programs which collect the activities of the DDL, DML, operating system, and the data base to provide the desired information.

Data Administrator

The group or individual who is assigned responsibility of the data base is known as the data administrator or data manager. In many cases, it is his duty to specify the schema of the whole data base and possibly subschemas for particular application programs. Other equally important functions are: 1) the preservation of the system integrity and security; 2) restructuring the data base to accommodate new record types; 3) monitoring system operations; and 4) defining procedures for restoring the data base in case of difficulties. Data administrators must rely on superior knowledge of the system, general knowledge of the data required by individual programs, statistics on usage of the data, and the required response time, to accomplish his ascribed functions and establish standards and procedures.

Storage Structures and Methods of Access

Storage structure and file structure are terms used to refer to the organization of individual files within the data base and relationships among them (3, 5, 7, 20, 25). Frequently the choice of file organization is dependent upon record structure, physical distribution of the records in the storage device, the indexing method employed for making an initial reference, and the manufacturer supplied file access methods. When selecting storage structures, the data base designer should build on the access packages available to him, if possible; thereby, avoiding duplication of the manufacturers work. Combining the above criteria the critical issue for file design becomes the efficiency of its performance. This must be established in terms of record storage, record retrieval, and management of the available storage space.[1]

Three common storage structures are sequential, indexed sequential, and direct structures. Accompanying the investigation of each of these structures is the means employed to access their data content. These methods are not necessarily the manufacturer-supplied access methods but they take advantage of the available access packages.

Sequential Structure

The best known and simplest of all storage structures is that which uses sequential organization. Records within this type of structure are stored in positions relative to other records according to a specified sequence. The ordered sequence is based on a common attribute of the records. If the attribute selected to order the records in the file is a data item within the record then the attribute is referred to as a key. The sequence of the records in a file may change by selecting a different key for a file and sorting on the basis of that key. Records within a sequential file are not required to have a key. When this occurs, records are stored as they enter the system so that the (N + 1)st entry follows the Nth entry. In the keyed and non-keyed cases the logical and physical order of records on the recording medium are identical.

The advantage of sequential organization is fast access per relationship during retrieval. Once a specific record has been

---

[1]Pan (20) offers an evaluation of several storage structures in "The Characterization of Data Management Systems."

retrieved, the next record in the data structure according to the relationship which was established by the key when the file was created is accessed rapidly. Other advantages of sequential structures are the ease of programming and the ability to store a large amount of data on a relatively inexpensive storage device, such as a tape volume.

Disadvantages appear while searching for a particular record in a large file and updating the sequential structure. Access of a specific key value would entail a complete scan of all records prior to the desired value, producing an undesirably slow retrieval time. Difficulties encountered in updating are revealed when inserting or deleting records in the storage structure. The former process requires that records in storage be split apart to make room for a new record. This process is reversed when deleting as existing records are compressed to fill the vacancy. Obviously, either process requires the moving or recopying of a large part of the file. As a result, sequential structures are more suitable for a tape-oriented data base with low volume.

Direct (Random)

To avoid the process of accessing every prior record when searching for a particular key value, the direct storage structure, storage device, and access methods were developed. These storage media are usually referred to as direct access devices and consist mainly of disk and drum units. In random storage structures records are stored and retrieved on the basis of a predictable relationship between the key of the record and the address of the location where the record is stored (3, 7, 9, 20). Having acquired the ability to access any record without regard to the previous access leads to the examination of three methods of accessing direct access devices. These are direct access, table look-up, and hashing.

Direct Access

Direct access methods require the programmer to know and supply the physical address of the record at storage and retrieval time. Such stringent requirements usually force a direct access device file to become device-dependent and location-dependent on that device. These features result in direct access methods being the least commonly used for direct storage structures.

Table Look-Up

In table look-up methods there exist an ordered pair-identifier/value--such that identifier is the record's key and value is the unique record address. When a record is stored or retrieved, the record key is matched with an identifier and the corresponding record address is used to access the record. By combining table look-up methods with data structures like inverted list and tree structures, an efficient access tool is established. For example, the indexed sequential structure (discussed later) employs a tree structured table to locate a record within the structure.

In addition to the desirable feature of rapid access to individual records, the table look-up method ensures that each record has a unique address. A third advantage of a key range scanning property is achieved if the identifier is maintained in some ordered sequence. This property is necessary for implementing the relational operators "greater than," "less than," etc. For example, assume there are 20 numeric identifiers in ascending order which are numbered 1 to 20. A request for all identifiers less than 10 is made. The first identifier is selected and determined to be less than 10, then all other identifiers are selected until the identifier is equal to 10. The results for this case would be identifiers one through nine.

Among the undesirable attributes of table look-up methods are: 1) an increase in the storage required, and 2) an increase in programming complexity in manipulating the table or directory. Suppression of these disadvantages is accomplished by accessing the directory with a binary search which is easy to implement and requires no additional link fields for the table. Another technique is to construct the directory using some type of tree organization with good search characteristics.[2]

Hashing

The last method of accessing a direct file is hashing or randomizing as it is sometimes called. Hashing decodes from natural language or coded input keywords to addresses by means of a mapping procedure (3, 7, 9, 12, 20). The coded representation of the term is a fixed length key falling within a specified range. This representation may represent an address of the record on a direct address storage device or it may represent the address of a location in a key

---

[2]The particular tree structure chosen for a directory would depend on the application. For example, if the application called for a rapid response with a small directory, an AVL tree may be appropriate. For the same requirements involving a large file a B-tree may be in order.

directory. If the key yields the address of a record on a DASD the record can be retrieved immediately. Otherwise additional processing similar to the table look-up procedure must be performed.

It is possible for the randomizing function to generate the same address from more than one record key. These occurrences are known as "collisions," or "synonyms." When collisions are encountered, algorithms must be devised to handle them. The chaining method is such a process. This method involves placing an additional link or pointer with each key. The link of the first record may now contain the address of the storage location housing the over-slow record. Another technique in manipulating collisions is to place the colliding entry into the next available empty location. A third procedure requires the hashing function to be repeated until an empty space is located. These are only a few of the techniques used to handle collisions. Various other methods are described in the open literature (9, 12).

The advantage of hashing functions is rapid retrieval of any record with a single access to the storage medium provided the randomization function chosen produces no collisions. Once synonyms begin to appear, the necessity of handling these records inhibits the system by causing an increased retrieval time and possibly causing an increase in required storage space.

<u>Regional</u> (PL/1 Direct Structures)

Programming language one (PL/1) allows the organization of direct storage structures in three ways: regional (1), regional (2), and regional (3) (27, 28). A major advantage of regional data sets over sequential and indexed sequential structures is that regional structures take full advantage of the characteristic of the direct access device. This permits the programmer to control the physical placement of records in the file and enables him to optimize the access time for a particular application. Each regional structure is divided into regions. Regions within a structure are numbered consecutively from zero and each may contain one or more than one record depending on the type of regional structure used.[3] Entries in the file are then accessed by a region number and possibly a key for a specific record. It is also permissible to access regional structures sequentially.

Selection of a regional structure is determined by the requirements of the particular application. Regional (1) structures are most suited to applications where there will be no duplicate region numbers, and where most of the regions will be filled (obviating wasted space in the data set). Regional (2) and regional (3) are more appropriate where records are identified by numbers that are thinly distributed over a wide range.

Indexed Sequential

An indexed sequential (IS) file is an ordered sequential file with indexes of record keys that permit rapid access to individual records as well as rapid sequential processing (7, 9, 12). The latter method is accomplished by arranging the logical records in the file in ascending key sequence. A specific record is obtained by searching

---

[3]The placement of records in a regional structure according to its relative position from the beginning of the file may be considered to be a particular type of mapping as discussed previously.

the index for the relative position of the record in the file and then a sequential search is employed until the desired record is located.

The obvious advantages of an indexed sequential organization to a data base system is that the file lends itself to being processed both sequentially and randomly. Since indexed sequential files are supported by most computer manufacturers, a second advantage of data independence is obtained for the data base system using this technique. IS files may be used as key directories for multilist files or possibly for inverted list files. An example of IS files being used as directories is the Shell Oil Company's technique. In this system there exist an owner-owned and owned-owner relationship among records. With Shell's technique the IS record contains two pointers--one pointing forwards to owned segments and the other pointing backwards to owner segments (24, 25).

With the many benefits of the indexed sequential organization it seems to be the ideal technique, except it exhibits several weaknesses. First, IS files use more space than random or ordinary sequential methods since the indexes must also be stored. Second, if the indexed sequential file is subject to frequent updates--insertions and deletions--significant increases in the time to retrieve a record may result. This time penalty leads to the major problem of reorganizing the IS file a time-consuming and costly process. Therefore, this technique is best associated with relatively static files.

Summary

The additional data base features are categorized into three groups: 1) data definitions, 2) storage structures, and 3) data administrators. Description of data in a data base is performed by the data definition process. Storage structures include the physical representation of data on a storage medium and means of accessing that data. Even though storage structures and data structures are considered as a separate data base feature, there exists a close relationship between them. In fact, this relationship must be taken into account for each particular application when designing a data base system. The final feature, yet the most important, is maintaining the data base. The person assigned this responsibility is known as the data administrator and his duties range from creation of the data base to ensuring its integrity.

CHAPTER V

DATA BASE FUNCTIONS

A generalized data base management system is potentially capable of providing generalized processing functions for either the programming user or for the non-programming user. The latter term indicates that the user is not required to write a program in a conventional sense. Therefore, users are being described according to what they have to do as opposed to what they have to be. The functions provided by a system for both users are called data manipulation facilities. These facilities vary from those supported by host language systems which provide explicit manipulation of a data base by a programmer to those supplied by the self-contained systems that provide implicit manipulation of a data base by the non-programmer. Subsequently the facilities contained in a system normally depend on the requirements of a particular application, it is not uncommon to find facilities for both categories of users. Presently there is no agreement on all of the generalized functions provided by the host language and self-contained systems. Although the most common functions deemed part of a generalized data base management system are file creation, file updating, file interrogation, and programming facilities (3, 4, 7, 16). Almost all systems possessing self-contained capabilities provide the functions of updating and interrogation. While all host language systems by definition provide a programming facility, they do not usually provide the self-contained functions of update and interrogation. There is no clear division with respect to creation between self-contained systems and host language systems.

Creation

Creation is defined as the process of making known to the data base management system a set of files on which the system can later perform other functions. This may be as simple as reserving space for the data set and adjusting or defining the data definition (schema) for the entire data base. On the other hand, it could imply the complete conversion of an existing file to an acceptable form for the purpose of creation or it may need to be programmed through the facilities provided by the data manipulation lanaguage.

Regardless of the method of creation the allocation of space and selection of device must be considered for each file. The space requirements can be estimated by either the data administrator from the existing entires (if applicable) and the expected rate of increase, or by the system. Likewise, the means required to access the data for updating and interrogation and response constraints of the data determine the choice of media type.

Monitoring of the creation cycle assures the integrity of the data base system. Reports generated through the various steps of the creation cycle should reflect any errors encountered in the process and statistics indicating the size of and resources used by files. If the creation function involves the transformation of an existing file to a workable form for the data base, then additional reports of validation errors and conversion statistics should be produced.

Updating

The update function is identified as the process of changing the value content of a data file. This includes modifying or deleting existing records, and inserting new records. Modifying an existing record is sometimes a process of deleting it and replacing it with its new form. Since updating may be done on all or part of the data file, executing any of the update functions requires that the descriptions both for the file to be changed and for the transaction (the update data) with which it is to be changed must exist.

Updating may be performed through an on-line or batch processing mode. In on-line maintenance the command and transaction are entered together and results in an immediate action by the updating facilities to carry out the designated function. The transactions in batch processing are queued with other update data and processed as one unit at some later time. The processing mode chosen for a particular system many times determines the user's control of the update procedure. For example, assume both the on-line and batch mode are supported. A user may only be allowed to modify selected items of an existing record in on-line mode, while the batch processing mode is reserved for the addition and deletion of an entire entry within the data base. The variations of user control in file updating and processing modes supplies an endless list. They are mentioned only for completeness.

Whenever information is to be written onto an existing data base file, there exists a possibility that the information does not conform to the definition established for that file. If this invalid data is allowed to enter the data base, then the integrity of the data base system is destroyed (as incorrect data could be retrieved). To avoid such accidental mishaps, at least, the following should be enacted: 1) The file being updated should be accessed only by the update routine during the update process; 2) The operating system or data base system should enforce the storage limits for the file; 3) All update functions should be preceded by some sort of validation facility (5, 13). These validation facilities should provide for extensive editing and possibly transformation of transactions before they are applied to the file; thus reducing the possibility of updating the data base with erroneous data. The validation process may be as simple as truncating zeroes or blanks and checking for character and numeric fields, or as complicated as satisfying several editing algorithms. Some systems even allow for logical relationships that must hold between transaction data and file data before a transaction can be processed.

Two facts should be mentioned about updating. First, the cost of updating the data base, if not monitored, may exceed its worth to the user. This is a prime reason for limiting when and how to update the files. Also a continuous increase in cost of updating may signify the need to reorganize the data base. The second fact is that updating is intrinsically a self-contained capability. This is because updating in self-contained systems implies a built-in processing algorithm in contrast to host language systems where the user programs his own updating.

Interrogation

The term interrogation is used to denote the process of extracting a specified subset of a data base and formating it for human uses or for later use by the system. The interrogation process consists of two parts: 1) the premise, and 2) the action.[1] The former defines how the part of the data base is to be selected (selection criterion). While the action defines how the operations of computation and formatting may be performed on the selected subset.

A premise consists of a logically connected set of conditions on one or more data items. A simple condition (also referred to as relational condition) is the smallest indivisible condition which may take various forms depending on the system. Simple conditions are constructed of three parts: subject, relational operator, and reference quantity (13, 16, 23). The latter two are often referred to as the predicate. The subject of a relational condition usually identifies a data item in the data base, while the basic relational operators are equals, not equals, greater than, less than, greater than or equal to, and less than or equal to. In some instances only a subset of these operators is permitted with a subject and in other cases they are expanded to encompass, and, or, not, nand (and-not), and nor (or-not). The third part, the reference quantity, may consist of a literal agreeing in type with the subject, another item identifier, or an arithmetic expression. Execution of the relational condition merely involves

---

[1]Lefkovitz (13) refers to these parts as data conditions and processing.

comparing the subject according to the relational operator with the reference quantity.

If the system has been expanded to include the logical connectives then very complex conditional expressions can be formed. The combination of simple conditions with logical connectives is known as compound conditions. These conditions are evaluated similarly to the relational condition with the exception of precedence rules for the logical connectives. Three frequently encountered precedence rules are: 1) left to right precedence; 2) "and" takes precedence; and 3) "or" takes precedence. Compound conditions also allow the establishment of a set of conditions on the same subject in the same expression. With such an occurrence the degree of repetition of the subject and relational operator differs among systems.

In addition to the compound expressions, a wide variety of special function conditions exist. Existence conditions are such a function. The existence condition checks the presence or absence of a value of an item (i.e., itemname blank, the condition is true if the item is blank otherwise the condition is false). Another common special function condition which allows relational conditions with respect to alphanumeric or string data is the scan condition. This function scans the subject to verify that it contains the literal string expressed in the condition. An added attraction of some scan functions is to allow one or more characters in the desired literal to be "don't care" characters which is interpreted as that position in the subject being any character.

Once the selection criterion has been established to be true, it is the responsibility of the action process to extract and format the data into reports or possibly into output files. If the system provides facilities for the user to design his own reports or files, then the user controls the action process. Otherwise, the system has control and reports (these may be selected by the user from a group of standard forms) or files are produced for the interrogation process.

Usually there exist a distinct difference between host language systems and self-contained systems in the handling of interrogation procedures. Self-contained systems require user specifications specifying what information he requires. Then a system process obtains that information in host language systems. The interrogation of the data base must be programmed in the conventional sense by the requestor. It is because of this difference in the handling of the premise action groups that interrogation is essentially a function of self-contained systems.

Programming Facilities

Programming facilities are considered features of host language systems. Therefore, access to any particular facility must be accomplished through a conventional programming language, which is termed the host language. The statement used to reference a programming facility must appear in one of three forms: 1) an explicit call with associated parameters, 2) a macro, or 3) some reserved word (verb) of the host language. The set of statements that form the programming facilities is named the Data Manipulation Language (DML). A DML is not a complete language and it must be embedded in some host language. DML statements are extensions to the host language which interface with the data base. This results in a mixture of host language statements and DML statements when writing application programs that access the data base. All calls to and from the data base to retrieve data, to add new data, to modify existing data or data relationships, and to delete existing data or data relationships are written with the DML statements. After this information has been extracted from the data base and placed in primary storage, it can be referenced and manipulated using the host language statements. Since programming facilities through the aid of a DML permit a more detailed and precise control over the information in the data base, an increased probability of destroying the integrity of the data base is evident. This means that a data base system which provides programming facilities, as described above, must be manipulated by a responsible programming user.

Summary

The functions provided by a system are called Data Manipulation Facilities. These facilities vary from those supported by host language systems which provide explicit manipulation of a data base by a programmer to those supplied by the self-contained systems that provide implicit manipulation of a data base by the non-programmer. The more common function of a generalized data base management system are file creation, file updating, file interrogation, and programming facilities. Systems possessing self-contained capabilities usually provide the functions of updating and interrogation. While all host language systems provide a programming facility, they do not usually provide the self-contained functions of update and interrogation. For the creation function there exist no clear division between self-contained systems and host language systems. The distinction above, between data base systems and the functions they provide, are often encountered when investigating data base systems, although it is not uncommon for these functions to overlap within a particular system.

CHAPTER VI

SUBJECT ANALYSIS

Although subject analysis is not necessarily a feature of GDMS, in many bibliographic data base systems subject analysis provides the initial contact between the user and information in the bibliographic data base. The subject description of a document starts by assigning to it a number of words that are to act as retrieval keys. These are called the keywords and they represent a condensation of the original text of the document, rather than its whole information content. They do not necessarily represent exactly the major topic of the text; they are a partial, approximate, imperfect guide to its information content.

Scanning a document to determine its subject content is the key operation in subject analysis. Logically two phases can be distinguished within the process: 1) scanning to select a set of words, phrases, or sentences that collectively represent the information content, and 2) deciding which of these are worth recording as being relevant to the interest of those who are expected to use the information system. If the analysis results in a short statement (or statements) concerning the content of the document then the result is known as an abstract of the document (13, 14, 23, 25). The process of forming an abstract is normally called abstracting. A second method of analysis is referred to as indexing. The result of this process is a set of descriptive terms for the document (13, 23, 25).

Abstracting and indexing serve two major functions for the user. First, they provide for current awareness which is a means whereby the user is kept in touch with the new work being published in his own and related subjects. Second, they are the major source for retrospective searching, both to locate individual articles and to compile bibliographies of a subject. The following discussion, labeled Indexing and Abstracting, will demonstrate how their makeup fits them for these tasks.

Abstracting

The abstract is a very brief statement in natural language of the essential content of the document. There are two types in general use today: informative abstract and indicative abstract (13, 23, 24). The former provides information and data that may be extracted directly from the document. In most cases, they contain particular values or results that are developed by the document and act in some degree as a substitute for the document. Rather than the explicit citation of facts, results, and conclusions of the document, the indicative abstract indicates what the document is about and what kind of information is contained in the document. Whichever method is chosen for abstracting it is generally agreed that an abstract should include: 1) the purpose of the item abstracted, together, perhaps, with its scope or magnitude; 2) the methods used, including equipment, materials, tests; 3) the results obtained, sometimes numerical data; and 4) the conclusion drawn (23).

The benefit of abstracting over indexing is that abstracts allow the user to obtain a better idea of the content of the document than does a list of descriptive words. But machine forming and searching of abstracts can be costly because of the more difficult programming and operating time involved.

Indexing

Indexing documents can be a purely mechanical operation of a highly intelligent one. It can be nothing more than a matter of listing words in the title of a document, or it can be a detailed intellectual analysis of its content. No matter how the work is done, the documents get labeled with a set of descriptive words (13, 23, 24).

There are two frequently used methods for forming indexes or document descriptors. They are 1) the assignation of certain keys, and 2) selective extraction. The latter is most often applied to subject words or phrases in the title, captions, headings, or the main text. Only a selection of these terms is extracted to form document descriptors. Establishing a selection criterion is usually related to word frequency and/or predicted user needs. The assignation of pre-existing keys involves the selective extraction of terms as before, but this must be followed by the transformation of terms into descriptors. In many retrieval systems this transformation process is left to the prerogative of the indexer, who must match his selected terms against a list of allowed descriptors.[1]

---

[1]A decision must be made when selecting descriptors in regard to word forms. That is, words such as differ, differs, and different may be treated as separate descriptors or a single term, called a stem, may be selected as the descriptor for these words. Stems usually are formed by designating several characters of a word to represent the word forms for that word--for the above words diffe may be selected as the stem.

The process of constructing indexes by machine is called automatic indexing. One method of automatic indexing is to have a human analyst compile a list of keywords of potential interest to future users of the index. This list is compared by the computer with each word in the text of a document; if a key word appears, the fact is recorded and these selected words make up the index entry for the document.

Another type of machine indexing works on an opposite principle: a human analyst compiles a list of words (referred to as a trivial word list) that are not to be selected for indexing. These include all the common "noise" words such as articles, pronouns, prepositions, and so on. Also included are general words that have little specific meaning (i.e., in scientific texts the words repost, theory, conclusion, etc., are too common to be useful as index entries). Each word of the text is then compared by the computer against the trivial word list and each word not appearing on the list is placed in the index. The construction of entries by the use of trivial words is much more common than by tagging significant words, since less intellectual effort is required at input.

KWIC (Keyword in Context) and KWOC (Keyword Out of Content) are two examples of automatic indexing employing either of the above methods for selecting keywords from the title entry of a document. The KWIC form selects the keyword and permutes the title so that the rest of the title wraps around on itself (refer to Figure 11). The KWOC form as the name implies actually (or logically) removes the selected keyword from the title (refer to Figure 11).

List of Possible Keywords:

Cyrstals, Ferroelectric, Thermodynamics

Title of Document:

"Thermodynamic Theory of Crystals with
Ferroelectric Properties."

Crystals with ferroelectric properties. Thermodynamic
theory of
ferroelectric properties. Thermodynamics theory of
crystals with
thermodynamics theory of crystals with fer-
roelectric properties.

(A) KWIC Indexing

Crystals
Ferroelectric
Thermodynamic

(B) KWOC Indexing

Figure 11. Automatic Indexing

Indexing and abstracting seem to be quite popular throughout the information retrieval world (8, 10, 14); yet there seems to be few answers to such questions as: the best way to select keywords, how many words should be selected from a document, and how to handle synonyms. Since the answers to these and many more questions are not available, the choice of how to achieve optimum performance is left to the designer, who must base his decision on the purpose of the system and on the amount of indexing labor that is available.

H. P. Luhn (14), who is considered to be the father of information retrieval, proposed an answer to the problem by performing both automatic indexing and automatic abstracting on the full text of the document by eliminating noise words and counting the non-noise words. The most frequently used words can act as descriptors or they can be compared with a list of approved descriptors, so that only approved words would be used. Automatic abstracting may also be performed by allowing the computer to print out four or five sentences from the text that have the greatest number of frequently used non-noise words (14, 24).

Summary

Subject analysis consists of scanning a document to determine its subject content. This process involves two phases: 1) selecting terms that represent the subject content, and 2) deciding which of these terms are relevant to the user. If the results of this analysis is a set of descriptive words (or keywords) assigned to the document then indexing has been performed. Although the way keywords are chosen differs, the main objective is to provide the user with the most descriptive terms and with the widest selection of terms pertaining to his interest. The two forms of automatic indexing are KWIC (Keyword in Content) and KWOC (Keyword Out of Content). Instead of indexing, the process of forming abstracts can be used for analyzing the subject content. This results in a short statement concerning the essential content of the document. If the abstract is a citation of facts, results and conclusions of the document then it is referred to as an informative abstract. On the other hand, if only an indication of what the document is about and what kind of information is contained in the document then an indicative atstract is formed. Often subject analysis provides the initial contact between the user and the document; therefore, when choosing a particular subject analysis scheme, the user's needs must be considered for each application.

CHAPTER VII

A BIBLIOGRAPHIC DATA BASE SYSTEM

When a researcher retrieves information from the available literature, he generally goes through three steps. First, he finds references to potentially appropriate documents; next, he obtains the documents; and finally, he searches them to locate the desired information. These three steps usually are called, respectively, reference retrieval, document retrieval, and information retrieval. This bibliographic data base is designed to be used mainly with the reference retrieval process.

The facilities provided for reference retrieval may be inconveniently located or available only during restricted periods and may be out-of-date or incomplete. They require time consuming manual search. The user may have to scan numerous nonrelevant references to locate relevant ones. In particular, satisfying several search criteria simultaneously frequently entails much unproductive work.

By developing an automated system that incorporates all the available literature into one centralized location, the unproductive work and time spent by the user may be substantially reduced. This computerized system is known as a bibliographic data base if it contains the following basic information: 1) author/title, which is concerned with topics relating to the origin of the document and linked with items serving to uniquely identify it, and 2) subject analysis, which is concerned with the information content of the document. The main purpose of a bibliographic data base system is to rapidly present the requester with only the relevant information pertaining to his topic of interest. Such systems normally provided a control language that interacts with the data base and the non-programming user.

This paper discribes a self-contained bibliographic data base system which is designed for the non-programmer who desires rapid retrieval of reference material. This data base system possesses self-contained capabilities as maintenance and interrogation of the data base is handled by pre-programmed processing algorithms which are invoked through a set of parameters available to the user. Presently the primary data contained for each document consist of three items: 1) the author, 2) the title, and 3) the location. To provide rapid access to documents and multiple entry points to the data base, this primary data is decomposed into five physically separate files (see Figure 12). These are the author file, location file, keyword file, inverted file, and the document file.

Also shown in Figure 12 (broken lines) are two "logical" files which exist within the keyword file. They (the thesis file and journal file) are maintained "separately" for special interest groups. Figure 13 illustrates the addition of an author resumé file and a document abstract file. Once this information becomes available and has been implemented, the user would be provided a more informative bibliographic data base system.

In Chapters II and III a full explanation of the features and functions of a data base system were stated. This outline will be employed in the following description of the implemented bibliographic

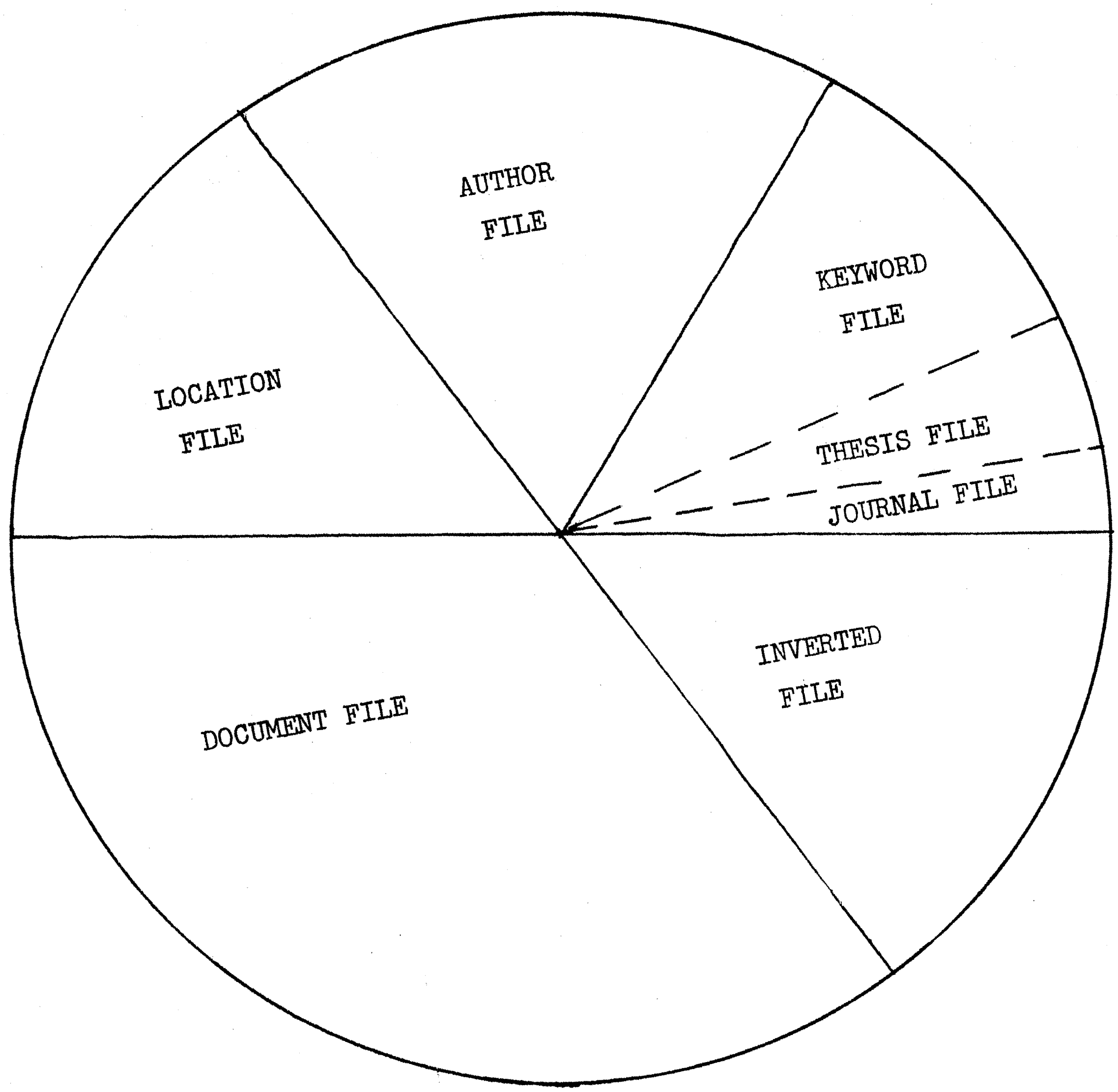


Figure 12. Present Data Base

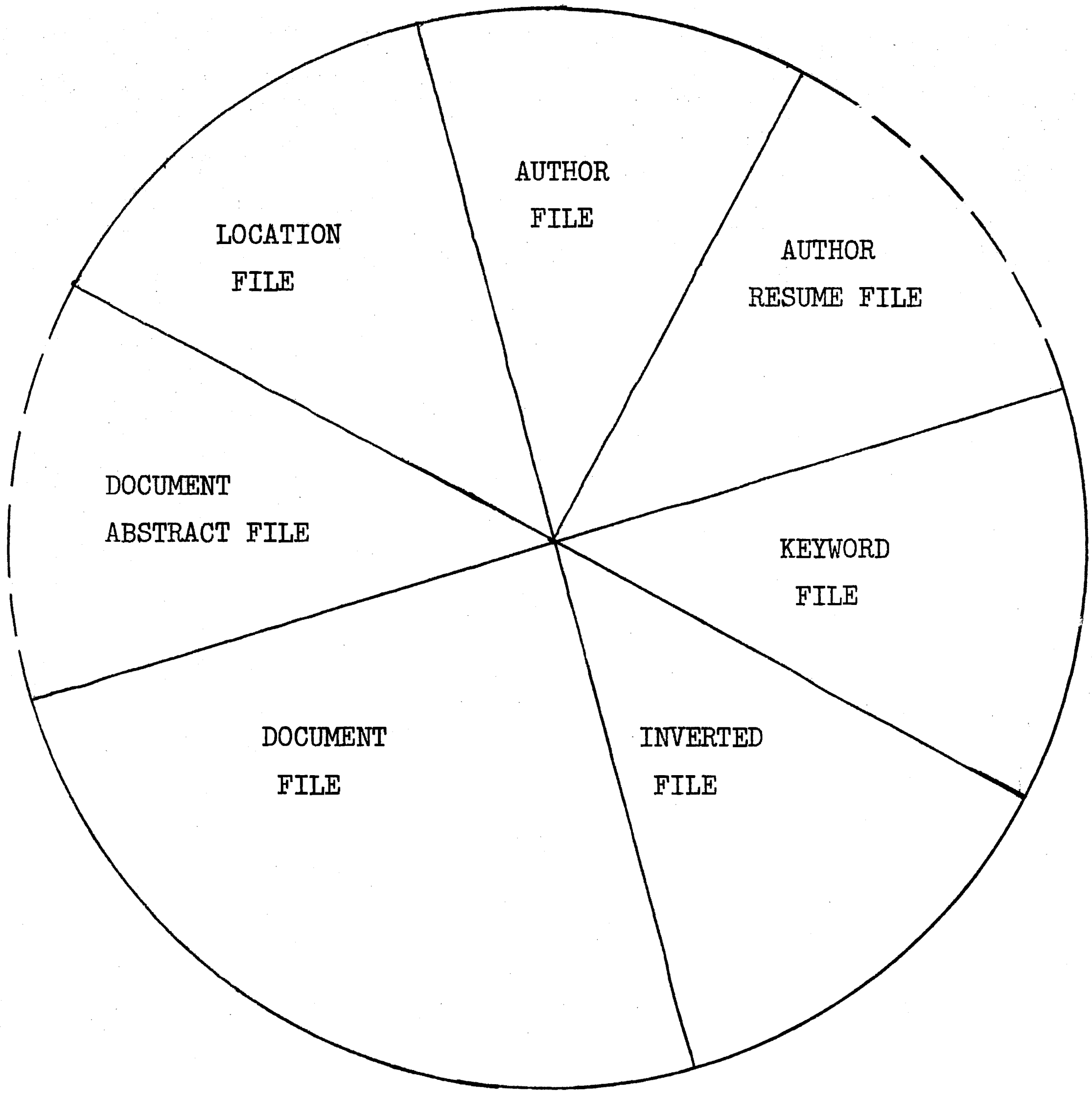


Figure 13. Possible Future Data Base

data base system. This entails a complete but general overview of the entire hierarchy of elements, information files, and processing algorithms that result in efficient management of information. Appendix D should be consulted for a more detailed study of the facilities, such as the algorithmic procedures, input and output formats, parameter formats, etc.

Data Administrator

There is one person assigned the responsibility of maintaining the data base. His duties consist of security and preservation of the system's integrity. The former is of minimum concern for this system because the entire data base is open to the users. The only security provided for the data base is password clearance for access to the system functions.

The system's integrity is the most important function of the data administrator. This involves the periodic updating of the data base. During the updating process, he will be responsible for visually checking all reports produced by the system and for recycling of all rejected entries. If any incorrect data has entered the system, such as misspelled names, incorrect locations, or nonrelevant keywords, the administrator may use the special purpose utility program (described later) to correct the mistakes.

Because the data administrator has superiority of the data base, it is his responsibility to provide optional features which are not available but would improve the user/data base interaction. This requires a moderate amount of programming on his part. The installation of the additional files, an author's resumé file and a document abstract file, would be performed by the administrator. This is the only situation in which a user, including the administrator, is not treated as a non-programmer who supplies parameters for the pre-programmed functions.

Data Structures

The statement that most data bases incorporate linked structures is supported completely by this data base implementation. In fact, every linked structure mentioned, except the B-tree, is used directly or indirectly within this system. By combining these structures, an interrelation between the files and their data is created. Due to these relationships, the user is able to access the data base through multiple entry points and still achieve a fast response for his request. Another feature of data bases that linked structures provide is the reduction of redundant data resulting in conservation of space.

The data base requirements for this bibliographic system include: an efficient means of maintaining a constantly changing bibliographic data base which may be accessed by author, location, and keywords; and although the system's input is processed in the batch mode, a relatively rapid retrieval time is required.

A survey of the advantages and disadvantages of the linked structures described in Chapter III indicate that the type structure supporting rapid retrieval of data and reasonable updating characteristics is the AVL tree. AVL trees also provide an ascending sequence of keys when retrieved by a postorder traversal. /See Knuth (11) p. 316/ Because the estimated size of the data base is from 1,000 to 3,000 documents, the entire AVL tree may be held in internal storage where the tree possesses good processing characteristics. By maintaining such structures for the author file, location file, and keyword file, the major requirement of multiple entry points for the data base is satisfied. Construction of a tree node is similar to the node depicted in Figure 8 (Chapter III) with the information field consisting of a key and a pointer to the record with that key. The selection of AVL trees and node construction combine to provide a self-organizing structure with rapid retrieval of data.[1] The inverted file which may only be referenced by the keyword file obviously consists of an inverted list structure. Each record of the inverted file contains four fields. The first three fields are pointers to records in the data base which correspond to the referenced keyword. While the fourth field is null or it points to the next inverted file record that is associated with the specified keyword; thereby, simulating variable length records. If an inverted list is maintained in sequence, the logical relations of union and intersection can be performed between keywords very easily. This property of ordering is inherent in the system, but is not required because only simple queries are allowed.

The last file, the document file, is by far the most important and elaborate in terms of the linked structures involved. Document records consist of the actual title of the document, an inverted list a doubly linked list, a linear linked list, and multilinked list. The

---

[1]Tomson (21) also employs AVL trees for directories in her description of an interrogation process.

primary reason for such a construction is to provide multiple entry points to the data base and to reduce the amount of redundant data. This file may be accessed only indirectly through the location file or through keyword file, but in return the document file may be used to access the author and location files. Access of these files through the document file is accomplished through the above-mentioned data structures. The inverted list consists of five links (this is a restriction of the system) with each link being null or pointing to one of five possible authors of the document. Since the links are always placed consecutively in the inverted list with any remaining links having the null value, the corresponding five links which make up part of the multilinked structure may contain the null symbol if it is the only document produced by that author or the link points to the next record in the multilinked list for the corresponding author (see Figure 14). If the title for a document is extremely long and requires an overflow area then a linear linked list is employed to associate the extra title record with the original document entry. The last data structure contained in the title records may be considered a doubly linked list which always points back to the location record for that document.

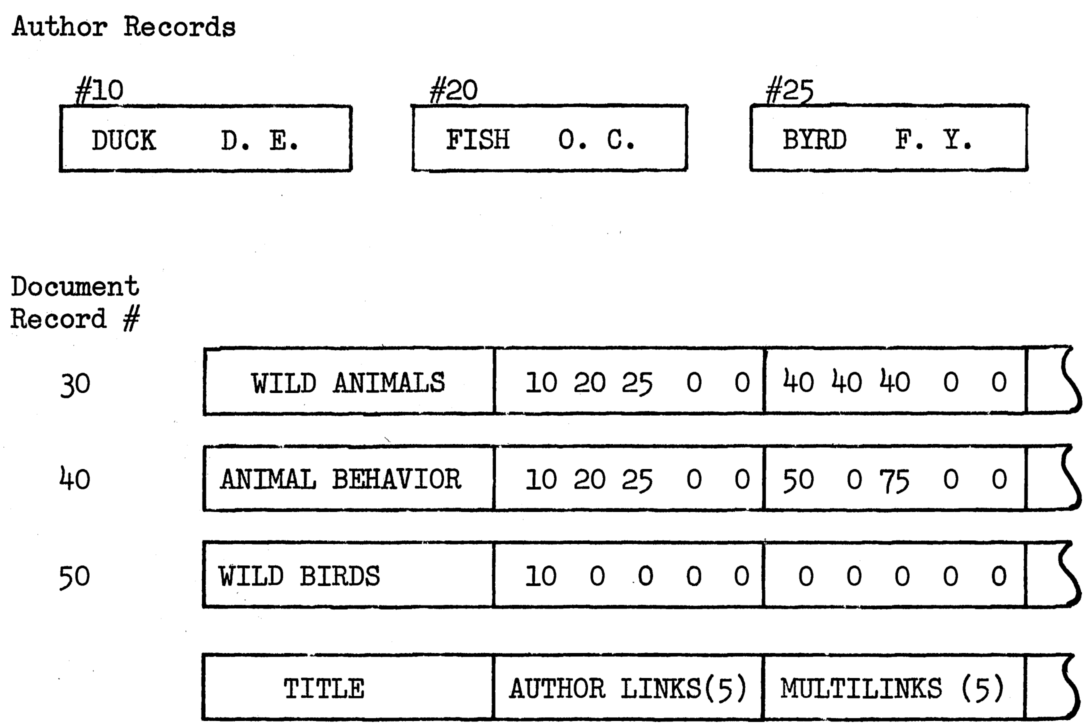


Figure 14. Inverted and Multilinked Structures in the Document File

Data Definitions

Data definition in Chapter II is defined in a strict sense for most self-contained systems and in a broad sense for host language systems. This strict definition of data applies to this bibliographic data base. Figure 15 (A), (B), and (C) describe the data definitions employed by the author file, the keyword file, and the keyword record respectively. Although the information contained in a field and the size of several fields differ the basic record configuration of each is identical. The fields comprising a record are: 1) a key field consisting of either an authors name, a document location, or a keyword; 2) a link to the information related to that key field; 3) a left link for the AVL tree; 4) a right link for the AVL tree, and 5) the balance tag for the AVL tree. The length of a field is measured in units of storage called bytes. Key fields for author records are 25 bytes long while the same fields for location and keyword records are 20 bytes in length. All link fields for these records comprise two bytes of storage and one byte is used to store the balance tag.

Figure 15 (D) depicts an inverted file record. These records consist of four fields. The first three fields are two bytes long and link the appropriate document file record with the keyword record. Field four is also two bytes long but links other inverted file records associated with the particular keyword.

The last definition is a document file record (Figure 15 (E)). It includes: 1) the documents title which is 140 bytes long, 2) a one byte link field for extra title records, 3) ten fields of two bytes used to link authors with the documents, and 4) a location link field two bytes long.

Storage Structures

The choice of storage structures which provide efficient performance for the data base system was based on the record structure, the physical distribution of the records in the storage device, the indexing method employed for making the initial reference, and the manufacturer-supplied access method.

Field (Bytes)	Description
1 - 25	Authors name
26 - 27	Link to location file
28 - 29	Left link for AVL tree
30 - 31	Right link for AVL tree
32	Balance tag for AVL tree

(A) Definition of Author Record

Field (Bytes)	Description
1 - 20	Location of document
21 - 22	Link to document file
23 - 24	Left link for AVL tree
25 - 26	Right link for AVL tree
27	Balance tag for AVL tree

(B) Location Record

Field (Bytes)	Description
1 - 20	Keyword for document
21 - 22	Link to document file
23 - 24	Left link for the AVL tree
25 - 26	Right link for the AVL tree
27	Balance tag for the AVL tree

(C) Keyword Record

Field (Bytes)	Description
1 - 2	Link to document file
3 - 4	Link to document file
5 - 6	Link to document file
7 - 8	Link to next inverted record

(D) Inverted Record

Figure 15. Data Definitions

Field (Bytes)	Description
1 - 140	Title of document
141 - 142	Link to extra title records
143 - 152	Five 2-byte links to author file
153 - 162	Five 2-byte links to document records with corresponding author
163 - 164	Link to location file

Figure 15. (Continued)

Since the author file, location file, and keyword file are usually held in main memory to achieve good processing characteristics, their file structure consists of a sequential organization. This makes updating simply an addition to the end of the file while the AVL tree manages the logical structures. However, if internal storage needs to be conserved only the file corresponding to the desired entry point needs to be in primary storage. The other two files will be processed as direct files on secondary storage devices with PL/1 regional (1) organization. This procedure will decrease retrieval time slightly but usually performance will not be degraded considerably.

To permit rapid direct access to the inverted file and document file a direct storage structure with regional (1) organization is constructed. The direct access is provided by a combination of the table look-up method and the BDAM (Basic Direct Access Method) package provided by the manufacturer. AVL trees act as directories which supply a relative record address and the BDAM package transforms it into the physical address of the record. These files may also be processed as sequential files. One such application is the scan function (discussed later) which processes the document file sequentially. When processing the files on secondary storage sequentially either the QSAM (Queued Sequential Access Method) package or a combination of the QSAM and BSAM (Basic Sequential Access Method) package is invoked.

Creation

Creation of the bibliographic data base is performed by a pre-programmed function. For this data base system the creation process is executed only once and this is at the initial construction of the data base.[2] The procedure progresses from a simple reservation of space and establishment of the file existance to the creation of each record in the file being built. Creation of the document file exhibits this less complex process. Its estimated space requirements are reserved on a direct access device and references to the file are created. Such a file which contains no data is normally called a null file. Presentation of the initial entries for a null file is by the first updating function which processes that file.

The remaining four files represent the more difficult process of the creation function. As stated earlier data structures usually provide an improved utilization of available storage space by maintaining an availability list. The author file, location file, and

---

[2]This excludes the process of re-creation or recopying of a file. For example, if the file (or files) needed to be restored from a back-up copy due to some malfunction within the system, the creation function would not be summoned. Instead it would require manufactured supplied utility programs.

keyword file use the right link of each tree node (see Figure 8) to construct a linear linked list. The inverted file supports the same linked structure but makes use of the last link of each entry. For this reason each entry of these four files must be composed and placed in its reserved storage location during the creation process.

The next function performed by the process is the creation of two logical files. They are made known to the system by the placement of the keywords "journal" and "thesis" in predetermined locations. These are the first and second records, respectively, of the keyword file. At this stage the logical files are known to the data base system and contain no data; therefore, they may be considered null files or more precisely as "logical" null files.

To ensure the performance of the creation process reports are produced indicating any errors encountered and statistics on the created files. It is the data administrator's function to verify this material to preserve the system's integrity.

Updating

Updating is defined as the process of changing the value content of a data file by modifying or deleting existing records or by inserting new records. To prevent destroying the system's integrity (1) only the updating programs may access the file during the update process; (2) the storage limits defined for each file are enforced by the operating system for the inverted and document file and by the data base system for the remaining files; (3) a validation facility is used to initiate the update process. With this self-contained system it is the function of the data administrator to perform the updating procedure periodically and correct any errors that may occur during the process. There are four built-in processing algorithms which allow the administrator to accomplish this task. Two of these are the edit and update functions. They perform the actual insertion of records and must be executed in sequence. The third algorithm is the print function which provides the administrator the ability to examine a subset of the data base or the entire data base. Utility functions are the last and most powerful functions. With these procedures the administrator is supplied the power of executing major modification to the data base.

The edit function is the first procedure to be executed when updating the data base. Input to the edit program consists of author cards (A), call numbers (N), or location cards, and title cards (T). Each item is punched one per card with the corresponding code letter appearing in column 72 (see Figure 16); the maximum number of cards for each type is 5, 1, and 6, respectively. Assuming that each entry passes the extensive validation process, one complete record is written on the journal file, if it is a journal, or the main output file. These two edited output files are sequential files and not considered part of the data base because they are merely temporary files. Verification of the edit function by the data administrator is accomplished by standardized reports produced during the process. These reports concern totals, valid entries, and invalid entries with their error messages. By passing parameters through the JCL (Job Control Language) the administrator is given a choice of what information he desires and how it is presented.

```
Column  1                                              72
   KEYS WILLIAM                                        | A |

   CASHMAN THOMAS                                      | A |

   001.64 K44B                                         | N |

   BASIC PRINCIPLES OF                                 | T |

   DATA PROCESSING                                     | T |
```

Figure 16. Typical Input Document

Immediately after executing the editing procedure, the update function is invoked. Input for this function consists of the two files produced by the validation process. At the start of the update program an entry is read from the main file. The author is removed from the entry and inserted into the author file. If it is a new author, it is added to the end of the file and the corresponding data structure is updated. An existing author is not added to the file but the linked list for that author is updated in the multilinked structure. The second item to be removed from the entry is the call number or location. If there exist a duplicate item in the location file, the entire entry is rejected and a new entry is read. Non-existing numbers are added to the end of the location file and the matching data structure is updated. By combining the title with the appropriate link fields a new entry is provided and the document file is updated sequentially. When the title exceeds the length allowed by an entry of the document file a new document record is constructed using the remaining portion of the title. Before adding this new record to the end of the document file, the corresponding linked list structure is updated. This process may be repeated only twice for any one document. Therefore, at most, three entries may appear in the document file for any particular document.

Subject analysis which is discussed in detail later is the last function to be performed on the entry. The analysis involves an auxiliary file and the title of the document. This additional file is an ordered sequential file consisting of undesirable keywords known as trivial words. The selection of keywords is performed as follows: A word (all alphabetic characters) is selected from the title; A binary search is used to determine if it is a trivial word; If so, then another word is selected from the title and the process repeated; Otherwise, the word is inserted into the keyword file (if not already there) and the process repeated until words in the title are exhausted. If an existing word in the document is to be disregarded as a keyword, it must match exactly with a word in the trivial word list. While for keywords only stems need to be equal for the word to be considered a duplicate keyword.[3] This allows forms of the same word to be placed together. When inserting words into the keyword file, a pointer is placed in the proper position in the inverted file and the word is added to the keyword file <u>only if there is not</u> a duplicate word (stem) in the keyword file.

The above process continues until every entry of the main file has been updated. At this point the journal file is then read as input so that the logical journal file of the data base may be updated. This procedure consists of updating the location file (if there exists a location), creating a document file entry and updating the document file, and placing the appropriate link in the inverted file. Although the steps are fewer than those for the main entries, they are completed in the same manner.

There are five reports produced by the update program (see Appendix F). Three reports contain a list of the newly added authors, locations, and keywords. Next is the report concerning errors during the update process and the last report provides statistics on the data base. Again it is the duty of the data administrator to check these reports and correct any errors. If at any time the administrator would care to investigate the data base further for possible discrepancies, he may use the print facilities. The print function will produce a list of the author file, location file, and keyword file in ascending sequence of keys or a report with the entire entry information associated with each key in the specified file. Also the complete

---

[3]This stem consists of the first five characters of a word.

contents of either the journal file or thesis file may be printed. By supplying the print program with parameters passed through the JCL, the administrator can produce the desired report or combination of reports.

Updating involves only insertions since the information represented in this data base system is considered to be permanent data. Based on this assumption of no deletion of entries within the data base, no deletion algorithm is provided. Even though there are no deletions, it is unrealistic to assume that no modifications to the data base are necessary. For this reason the data manager is provided an utility function which makes available certain desired modifications.

Change and delete are the two modifications permitted by the utility program. The latter may be used only with the keyword file. It is invoked when the administrator supplies the utility function with the "delete" command and the keyword to be deleted. This gives the data base manager the power to remove words from the keyword file that have been determined to be trivial words. To correct errors such as misspelled author names and keywords, or an incorrect document location, the data base manager may use the "change" command. When requesting a change these additional parameters must be supplied: 1) a code of A, C, or K for author, call number, and keyword, respectively; 2) the "old" word that is the incorrect word; and 3) the "new" word which consists of the correct word. One of two algorithms is called upon when a change modification is summoned. If the new word does not exist in the appropriate file then there is a direct replacement of the old word with the new word. This one to one correspondence means that no additional space is used and none is freed. The second algorithm is executed when the new word is a duplicate of a word in the file being considered. This operation involves a merge between the list related to the old word with the list of the new word. Space containing the old word will be freed and it may be reused. Management of this freed space by the change (or delete) operation is handled by the data base system through the availability lists discussed earlier.

Interrogation

The self-contained capability of interrogation is provided by a built-in processing algorithm. Only the conditional relation of equal to is permitted to be used by the interrogator.[4] Therefore, it is implicity specified for all requested interrogations. A set of user commands exist for the interrogation process and the use of a command with its required parameters will be called a simple query.[5] Allowable commands that the user may specify are A, C, K, S, and KS. The first three indicate the file which will be used as the entry point into the date base. They are the author location, and keyword files, respectively. Entering the data base through the author file will produce for the user a list of all material contained in the data base that was published by the particular author. A similar procedure is followed when interrogating through the keyword file except the list produced contains all documents in the data base whose title contains

---

[4]See User's Guide (Appendix B) for further information on interrogating the bibliographic data base.

[5]A thesis by Thomson (21) investigates some of the problems involved when using more complex queries during the interrogation process. A discussion of these problems can also be found in Knuth (12).

the specified keyword. If the command specifies the location file for searching the data base then only one item, specifically the item with that location, is contained in the report. Supplied with the S command, which is the mnemonic for the scan function, is a character string not exceeding thirty characters. This function will sequentially process the entire document file constructing a report of all documents whose title contains the desired literal string. The last operation available to the user is the keyword/scan (KS) command. When using this option the requester enters the data base through the keyword file and processing is identical to the K command. But if the desired keyword does not exist in the keyword file, the scan function is invoked using the particular keyword as the searching literal. An appropriate message is produced for any query in which the desired information is not contained in the data base.

Subject Analysis

The purpose of subject analysis for the implemented data base was to allow the individual user fast access to material on computer science topics which relate to his interest. It was stated earlier that no canonical forms which achieve some optimum performance for indexing and abstracting schemes have been established. In fact, the only criterion suggested for a specific design was one of economics and satisfaction of the system's purpose. It is in accordance with these ideas that the particular type of indexing and method of keyword selection were chosen for use in this system.

Due to the volume of information composing the bibliographic data base, the cost and time involved in obtaining abstracts or even table of contents could not be justified at this time. This provided the data base with only the essential information of the existing documents. These are the location, author, and title of the document. The combination of the above factors determined an inplicit assumption for the analysis. The assumption stated explicitly is that the title of the documents represented the true content of the material. Enforcement of the assumption forces the inclusion of an indexing scheme and the exclusion of an abstracting scheme. However, if abstracts for the documents should become available, they could very easily be incorporated into the data base by the data base administrator (see Chapter VI). This addition would provide a more complete and informative system for the user.

The following information was considered when deciding which type of indexing to install, KWIC or KWOC. The KWIC index increases both processing time and storage space, and it is more difficult to program. KWIC indexes are probably more suited for examining the entire data base or a large subset of it. The opposite may be stated for KWOC index; they are easier to program, and less extravagant with time and space (26). The comparison of the methods and the system's purpose indicates the logical choice of KWOC indexing.

The construction of the index words for the KWOC system is accomplished by selective extraction involving a list of trivial words matched against words in the title of the document. When a word from the title does not match with the trivial word list, the word is used to update the keyword directory.[6] Because all possible queries

---

[6]For further information on selection of keywords and on the updating procedure, see Appendix C.

against the data base could not be anticipated, this method presents the user with the widest selection of indexes and the opportunity to locate material on minor, as well as, major topics.

Summary

A bibliographic data base contains information concerned with topics relating to the origin of a document linked with items serving to uniquely identify the document. The implemented bibliographic data base system is a self-contained data base system which is designed for the non-programmer who desires rapid retrieval of reference material. This data base system possesses self-contained capabilities as maintenance and interrogation of the data base is handled by pre-programmed processing algorithms which are invoked through a set of parameters available to the user. Presently the primary data contained for each document consist of three items: 1) the author, 2) the title, and 3) the location. To provide rapid access to documents and multiple entry points to the data base, this primary data is decomposed into five physically separate files. These are the author file, location file, keyword file, inverted file, and the document file. Two additional files are the journal file and the thesis file. These files are logical files which are provided for special interest groups and they exist within the keyword file.

CHAPTER VIII

SUMMARY AND RECOMMENDATIONS

This paper has attempted to give a broad description of generalized data base management systems. Such a description requires examination of both the logical and physical representation of information in the data base, plus the means of accessing that data. Maintenance of the data base and manipulation of data by the user are the final topics discussed in this investigation. It is these two latter functions that provide the grouping of data base management systems into host language systems and self-contained systems.

No matter which technique or combination of techniques are used to represent the information and to define the user's capabilities, the ultimate goal of a data base system is to provide an efficient means of handling information for various applications. It is this idea that leads to the selection of specific data and storage structures and determines the user's interface for a bibliographic data base system. Since this particular data base system supports many of the known data management techniques it may seem too complex for the novice user. To avoid such confusion, the information retrieval techniques are transparent to the user. In fact, the user is treated as one who gives a command and the data base system executes the orders. Thus, the implemented data base system provides the user with a centralized location and a current awareness of material available to him on a particular subject matter, namely computer science.

In most practical applications the user's needs can never be fully anticipated. Therefore, as time progresses this normally gives rise to an increasing amount of information needing to be stored within the data base and to additional requirements for the system. The existing system should be able to handle the growth with few or no modifications. Two such expansions, of a document abstract file and an author resumé file, were mentioned earlier. All of this information could be added to the existing bibliographic data base with very little effort.

For the author resumé file this would involve: 1) creation of a PL/1 regional (1) file and placing the resumés of the existing authors into this file in the same sequential order as its corresponding author in the author file, 2) adding a statement to write the resumé onto the resumé file each time a new author is added to the author file, and 3) adding a statement to print the information for the user. The first step is performed only once and it is necessary only because the information was not available earlier. Step two implies the important fact that no additional space for links is needed. The reason for this is that each author's name exists only once in the data base and the name is in the author file. An identical procedure would be followed for the addition of the document abstracts except the implied position of the abstracts in the abstract file would be determined by the location file since there exists one location for every document.

A third recommendation is to expand the facilities of the interrogation process to allow the user to employ Boolean and range queries when using keyword referencing. If a stipulation, such as allowing only one Boolean relation (and, or, not) per query, is enforced then the procedure described in Chapter IV for the inverted file could be implemented with a moderate amount of effort. On the other hand, if queries are allowed to contain any number of Boolean relations and possibly mixed with simple and range queries involving the author file, location file, and keyword file, then a substantial amount of effort may be involved. A major part of the extra effort would involve: 1) constructing a routine to decode the Boolean query (possibly optimizing the request by constructing an equivalent query that is more efficient in processing the request), and 2) designing a method to handle intermediate results.[1]

Although the implemented data base system performs all functions correctly and fulfills the user's requirements it is possible in the distant future to exceed the capacity of the system with an enormous increase in document information. This is due to a characteristic of the system that requires an AVL tree to be maintained in primary storage (memory). One approach to solving the difficulty is to install a paging scheme in which a single segment of the tree is contained in a single page (see Knuth (11)). A procedure of this nature could lead to a major problem of constantly bringing pages in and out of memory, thus degrading the performance of the entire system.[2] A more desirable

---

[1] In Tomson's (21) paper AVL trees are used as directories and as storage of intermediate results. Also, Knuth (11) gives a discussion of processing such queries. Palermo (19) describes several techniques for decoding queries to optimize retrieval of information.

[2] In paging environments this problem is known as thrashing.

solution would be to convert the AVL trees to B-trees which are better suited to large files maintained in secondary storage. If the basic node structure of a key and pointer to the record is kept then the conversion should occur with a minimum amount of difficulty. It would entail the reorganization of the records in the author file, location file, and keyword file, and slight modifications of existing programs. The programs would be modified by merely replacing the AVL tree routines with B-tree routines. Subsequently, the systems algorithms are untouched.

Due to the rapid increase in the volume of data needing to be maintained and processed and the versatility offered by the data base management systems, it seems likely that the latter will be useful in the field of information storage and retrieval in the foreseeable future.

SELECTED BIBLIOGRAPHY

(1) Atherton, Pauline. "Bibliographic Data Bases--Their Effect on User Interface Design in Interactive Retrieval Systems." In Interactive Bibliographic Search: The User/Computer Interface. New York: AFIPS Press, 1971, 215-23.

(2) Bachman, C. W. and S. B. Williams. "A General Purpose Programming System for Random Access Memories." AFIPS Fall Joint Computer Conference, 1964, 411-422.

(3) Brynes, C. J. and D. B. Steig. "File Management Systems: A Current Summary." Datamation, 15, 11 (November, 1969), 138-142.

(4) Codasyl Data Base Task Group. October 69 Report. New York: Association for Computing Machinery, 1970.

(5) Codasyl Systems Committee. Features Analysis of Data Base Management Systems. New York: Association for Computing Machinery, 1971.

(6) Davis, William S. "Empirical Behavior of B-Trees." (unpub. Masters thesis, Oklahoma State University, 1974.)

(7) Dodd, George G. "Elements of Data Management." Computing Surveys, 1, 2 (June, 1969), 117-133.

(8) Fischer, M. "The KWIC Index Concept: A Retrospective View." American Documentation, 17, 3 (April, 1966), 57-70.

(9) Gitomer, J. H. "Data Base Concepts and Considerations." In Handbook of Data Processing Management, Vol. 5. New York: Auerbach, 1971, 47-73.

(10) Jordan, John R. "Let the Computer Select Your Reading List." Datamation, 8, 2 (February, 1970), 91-94.

(11) Knuth, D. E. The Art of Computer Programming, Vol. 1. Reading: Addison-Wesley, 1969.

(12) Knuth, D. E. The Art of Computer Programming, Vol 3. Reading: Addison-Wesley, 1973.

(13) Lefkovitz, David. File Structures for On-Line Systems. New York: Spartan Books, 1969.

(14) Luhn, H. P. "The Automatic Creation of Literature Abstracts." In Key Papers in Information Science. Washington, D.C.: The American Society for Information Science, 1971, 87-94.

(15) Lyon, J. K. An Introduction to Data Base Design. New York: Wiley-Interscience, 1971.

(16) Minker, John, "Generalized Data Management Systems: Perspectives, Dictionary and Tree Searching." 1971 International Seminar on Information Storage and Retrieval, 1971, 1-222.

(17) Nievergelt, J. "Binary Search Trees and File Organization." Computing Surveys, 6, 3 (September, 1974), 195-207.

(18) Olle, T. W. "A Comparison Between Generalized Data Base Management Systems and Interactive Bibliographic Systems." In Interactive Bibliographic Search: The User/Computer Interface. New York: AFIPS Press, 1971, 203-214.

(19) Palermo, Frank P. "A Data Base Search Problem." In Information Systems COINS IV. New York: Plenum Press, 1974, 67-100.

(20) Pan, George S. "The Characterization of Data Management Systems." Data Management, 9, 6 (June, 1971), 18-23.

(21) Tomson, Harriet, "An Information Storage and Retrieval System Using AVL Trees." (unpub. Masters thesis, Oklahoma State University, 1973.)

(22) Van Doren, J. R. and J. L. Gray. "An Algorithm for Maintaining Dynamic AVL Trees." In Information Systems, Vol. 4. New York: Plenum Press, 1974, 161-180.

(23) Vickery, B. C. Techniques of Information Retrieval. Connecticut: Archon Books, 1970.

(24) "Creating the Corporate Data Base." EDP Analyzer, 8, 2 (February, 1970).

(25) "Organizing the Corporate Data Base." EDP Analyzer, 8, 3 (March, 1970).

(26) "Processing the Corporate Data Base." EDP Analyzer, 8, 4 (April, 1970).

(27) IBM System/360 Operating System, PL/1 (F) Language Reference Manual, (GC28-8201). New York: International Business Machines Corporation, 1972, 137-142.

(28) IBM System/360 Operating System, PL/1 (F) Programmer's Guide, (GC28-6594). New York: International Business Machines Corporation, 1972, 144-158.

APPENDIX A

GLOSSARY

Abstracting - A short statement (or statements) concerning the content of the document.

Bibliographic Data Base - A data base that continas information concerned with topics relating to the origin of a document linked with items serving to uniquely identify the document.

Data Administrator - Person responsible for maintaining the data base.

Data Base - A set of one or more files containing nonredundant data and interrelated data items which are processable by one or more applications.

Data Base System - Is a system composed of a data base and of the facilities provided to manipulate the data base.

Data Definition - A description of the data in a data base.

Data Structure - Logical structures for representing the data in a data base.

Host Language System - A data base system that provides user capabilities by augmenting a general purpose language.

Indexing - Process of labeling a document with a set of descriptive terms.

KWIC - Keyword in context

KWOC - Keyword out of context

Self-Contained Systems - A data base system that provides user capabilities by executing pre-programmed algorithms.

Storage Structure - Physical representation of the data on a storage medium and means of accessing the data.

Subject Analysis - Scanning a document to determine its subject content.

APPENDIX B

DEFINITION OF THE BIBLIOGRAPHIC DATA BASE SYSTEM

The definition of the bibliographic data base may be stated formally as follows:

> Design and implement a bibliographic data base system using the PL/1 programming language. This system should provide all necessary information pertaining to a user-supplied author or location (usually a call number). In addition the system should provide an interactive search facility of the entire data base for user-supplied keywords.
>
> Presently, the input data for the data base will consist of punched cards containing the author, location, and title of each document relevant to the data base.

APPENDIX C

USER'S GUIDE

The implemented bibliographic data base system provides the user with a centralized location for obtaining information on computer science topics. The data base system will provide all data contained in the data base pertaining to the specific item requested. Examination of the data base is through the use of control information obtainable from the data base administrator and of commands specified by the user. Any number of commands may be given within one interrogation process but each command must be a single card. Allowable commands are A, C, K, S, and KS. All commands start in column one and they are followed in column ten by the item to be located.

Author (A) commands specify that the item being passed is an author's name. The command card must appear in the following form: 1) last name first (not to exceed 25 characters), 2) one blank, and 3) the author's first initial. If the author exists in the data base, all information pertaining to him will be printed.[1]

Locate (C) commands initiate a search for the location of a document. The item usually takes the form of a call number and may not exceed 20 characters. If the item exists, the document and all related information with that location is printed.

Keyword (K) commands indicate that item is a keyword which does not exceed 20 characters. The keyword specified should be the singular form of the keyword.[2] All documents whose title contains this keyword

---

[1]There is one special "author" named "GENERAL A" which contains material that has no specific author such as reports on symposiums, and conference proceedings, etc.

[2]Since stems are used for comparison, there is no need to distinguish between similar words, such as computer and computers.

will be printed for the requestor, if this is a valid keyword.[3]

Scan (S) commands provide the user with a means of searching the titles of all documents in the data base for a specific item. This item may contain any allowable alphanumeric character but may not exceed 30 characters. Output consists of all documents containing the literal in their titles.

Keyword/scan (KS) commands combine the K command to aid the user in his search. A normal K command is executed by the system. If the keyword is found, all information is printed and the next command is read. When the keyword does not exist, a scan command is issued by the data base system using the keyword as the literal to be located.

Illustrated below (Figure 17) is several commands that would invoke the interrogation process. Notice that the order and number of commands is irrelevant to the process. Although out from the process concerning the success or failure of a command is produced in the same sequential order that the commands are read.

---

[3]There exist two special keywords called "journal" and "thesis." The former will produce all available journals. The keyword thesis prints all theses contained in the data base.

Column	1	10
	A	AUTHORɃN
	S	INFORMATION RETRIEVAL
	K	COMPUTER
	L	458.32 LC12
	K	GRAMMAR
	KS	COBOL

Ƀ = BLANK

Figure 17. Sample Input

APPENDIX D

DATA ADMINISTRATOR'S GUIDE

Create

There exist no external input to the creation procedure. Output for the process consist of a list for each file created and of the maximum number of possible records.

Program Create reserves space for all files on a direct access device. The document file, which is a PL/1 regional (1) data set, is opened and closed according to specifications listed in IBM Manuals (20, 21). The author file, location file, and keyword file use the right link of each tree node (each node is a record) to construct a singly linked list. This list provides the next available node when necessary. An identical process of linking each node with the next node is performed with the last field of each record in the inverted file. Two logical files are created within the keyword file. These files are established by placing the keywords, journal and thesis, in position one and two of the keyword file (the AVL tree for the keyword file is also updated). The final step in the process is to initialize all counters used to maintain the system. Since record zero of all files is not used for file information, it is used for system information (mainly counters). The particular file records containing this information are chosen to reduce extra read statement. The placement of counters is depicted in Figure 18.

Inverted Node	LINK	LINK	LINK	LINK	
Tree Node	KEY	LINK	LL	RL	BT
Record Zero of:					
Author File		AV		RT	
Location File		AV	TAV	RT	
Keyword File		AV	IAV	RT	
Inverted File	ACNT	CCNT	KENT		

AV - Next available node for that tree
IAV - Next available record for inverted file
RT - Root for that tree
TAV - Next available record for the document file
ACNT - Count of author records
CCNT - Count of location records
KCNT - Count of keyword records

Figure 18. System Count Locations

Edit

Input for the edit program consist of new documents to be added to the data base. The document information is punched in 80-column cards (see Figure 16). All cards must contain an A, N, or T punched in column 72 for Author, Location, or title respectively. Columns 73-77 will contain entry numbers while columns 78-80 contain sequence within entry. Only one item is placed on a card and it is assumed that column 1 is the first position of the item.

Locations cards may not exceed 25 characters in length and they must be the first card of an entry.

It is recommended that author cards follow the location card for the entry but this is not a requirement. There exists one author per card with last name first and at least the author's first initial. The last name may not exceed 25 positions. Separation of the last name and the first name or initial is by one or more blanks or a comma.

Document titles are placed in column 1 through column 70 of the title card. There may be up to six title cards per document with column 1 of the successor card following column 70 of the predecessor card.

The remaining input for the edit program is through the "PARM" parameter on the execute card in the JCL. This number specifies what information the administrator desires and how it is to be presented. Allowable digits are one, two, or three with one being the default. If a one is passed to the program, then all good records are written first followed by a list of error messages and rejected entries. Two specifies that all records are to be printed sequentially with rejected entries marked by asterisks. Also a list of rejected entries is produced. This parameter displays the error entry and the error message, thus enabling the administrator to easily pinpoint the mistake. When a three is used only the essential information of rejected entries with error messages is printed. Totals regarding the number of journals and records added and the number of errors encountered are produced regardless of print options.

The edit program is the first procedure to be executed when updating the data base. This process begins by reading an entry and determining the entries validity. Validation of the entry is performed in three parts. First the location is checked for errors. Next, the author (or authors) is checked and last, the title is verified. If the entry is valid and it is a journal, then it is placed on the journal file or else it is placed on the main output file. An invalid entry is marked as such and rejected. This placement or rejection of an entry is performed each time a new entry is read.

If an entry exists with all correct information except an author, then an author of GENERAL A is created. This has been implemented for items such as conferences, symposiums, etc.

Illustrated in Figure 19 is the format of the two output files created by the edit process. They are only temporary files and they are destroyed after execution of the update program. Since only 140 characters are allowed in one record and the maximum for an entry is 420 characters, a flag is used to denote the next record as containing more title information.

Field (Bytes)	Description
1 - 25	Location of document
26 - 160	5 Fields of 27 characters each containing an author
161 - 300	Title of the document
301 - 302	Flag for extra titles
303 - 307	Entry number

Figure 19. Edit Files

Update

Input for the updating algorithm consist solely of the two files created by the edit program. The output is a standard report containing a sequential list of all new authors, new locations, and new keywords added. This is followed by a summary of entries in the data base.

First, the updating procedure will read the existing files (the author file, location file, and keyword file) into memory and store all count fields. Then processing of an entry begins by reading a record from the main input file (this is the main output file produced by the edit program). The procedure to update an entry is as follows. The author is removed from the entry and inserted into the author file. If it is a new author, it is added to the end of the file and the corresponding data structure is updated. The link field for that author record is set to point to the record in the location file where the location of that document will be placed. An existing author is not added to the file but the linked list for that author is updated in the multilinked structure. The second item to be removed from the entry is the call number or location. If there exists a duplicate item in the location file, the entire entry is rejected and a new entry is read. Non-existing numbers are added to the end of the location file and the matching data structure is updated. By combining the title with the appropriate link fields a new entry is provided and the document file is updated sequentially. These link fields are: 1) a backward reference to the location file, 2) a backward reference to each author, 3) the corresponding multilinks for each author, and 4) a pointer to extra title records. When the title exceeds the length allowed by an entry of the document file, a new document record is constructed using the remaining portion of the title. Before adding this new record to the end of the document file, the corresponding linked list structure is updated. This process may be repeated only twice for any document. Therefore, at most three entries may appear in the document file for any particular document.

Subject analysis is performed on the entire title of the document. The analysis involves an auxiliary file and the title of the document. This additional file is an ordered sequential file consisting of undesirable keywords known as trivial words. The selection of keywords is performed as follows: a word that must be all alphabetic characters is selected from the title; a binary search is used to determine if it is a trivial word; if so, then another word is selected from the title and the process repeated until all words in the title are exhausted. If an existing word in the document is to be disregarded as a keyword, it must match exactly with a word in the trivial word list. While for keywords only stems need to be equal for the word to be considered a duplicate keyword. The stem consists of the first five characters of a word. This allows forms of the same word to be placed together. When inserting words into the keyword file, a pointer is placed in the proper position in the inverted file and the word is added to the keyword file only if there is not a duplicate word (stem) in the keyword file.

The above process continues until every entry of the main file has been updated. At this point the journal file is then read as input so that logical journal file of the data base may be updated. This procedure consists of updating the location file (if there exists a location), creating a document file entry and updating the document file, and placing the appropriate link in the inverted file. Although the steps are fewer than those for the main entries, they are completed in the same manner.

Utility

Input in the form of command cards control the execution of the utility process (see Figure 20). The two legal commands are "change" and "delete." One command is executed per command card with the command word starting in column one. This is followed by one blank then the character A, C, or K. These characters indicate the file to be corrected. With command change the key expression "OLD=" must precede the item in error which is placed in quotes immediately following the key expression. The replacement item for change commands is also in quotes and it is preceded by the key expression "NEW=."
Command change is used to correct misspelled keywords and author

names and to replace old locations with new locations.

```
1                                                          80
CHANGE A             OLD="ERROR"          NEW="CORRECTION"

1                                                          80
DELETE K        KEY="TRIVIAL"
```

Figure 20. Typical Utility Input

The character K is the only allowable letter that may follow a delete command since deletion is only performed on the keyword file. "KEY=" is the key expression which precedes the keyword to be deleted. This keyword is also in quotes.

A command is read by the utility program. If it is a change command, the file to be changed is determined by the character following the command. If this character is a C, then the location file is to be altered. This is performed by deleting the old location specified and inserting the new location. This new location cannot be a duplicate of an existing location (an error message is produced if this occurs and program execution stops). If an author is to be changed, then the old name is deleted and the new name is inserted. If no duplicate name exists for this new name then this process is finished. Otherwise, it is a duplicate name, and the list of documents related to the old name must be added to the list of documents for the duplicate name. The last possible file to be changed by a change command is the keyword file. Again, the old word is deleted and the new word is inserted. If a duplicate word exists for the new word, then the inverted records for the old word must be added to the inverted list of the duplicate word. In either case of a duplicate word or a new word, the corresponding word is located in the title and corrected. During this correction process involving the document titles it is possible to add a new title record to hold the expanded record. Otherwise there is space available in the document record--words in the title record are separated by one blank within a document record; so each search for exapnsion space is within one document record. If expansion space is needed but none is found, the overflow characters for that record are stored; then they are concatenated in front of the next document record for that document. If no extra document records exist, then a new title record is built.

If the delete command is specified only for the keyword file, this is a simple process of deleting the keyword file. All freed space, such as the keyword record and the inverted file record, is returned to the availability list (this also occurs when space is freed by the change command). The above process continues until all commands have been processed.

Print

Input to the print program is through the "PARM" parameter on the execute card in the JCL. The parameters are A, C, J, K, and T. These represent authors, locations, journals, keywords, and thesis, respectively. Each command, except K, will produce a report in ascending order of command field of all information in the data base. Command K will only produce a listing of all keywords in ascending order. If only a listing of authors or locations is desired, then the command should be preceded by an L such as LA or LC. Although the commands may be in any order, they will only be processed once. Figure 21 demonstrates the use of the PARM parameter for print.

```
// EXEC ..., PARM.GO="ACJKT"
```

```
// EXEC ..., PARM.GO="LCLA"
```

Figure 21. Print Input

Program print decodes the parameters specified and produces the requested input. If the reports require the author file, location file, or keyword file, then a postorder traversal is used to obtain the information (see Reference 4). If the logical files of journal or thesis are specified, then the appropriate inverted file records are traced producing the desired information.

Interrogation

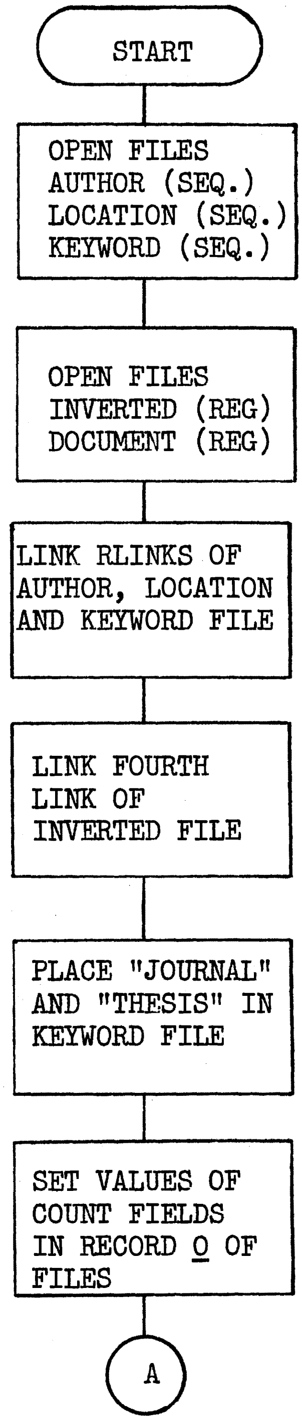
Interrogation input consist of command cards with one command per card. The commands are punched starting in column 1 with the search item beginning in column 10. Existing commands are A, C, K, S, and KS. A, C, and K specify which file is to be searched (i.e, author, location, and keyword file, respectively). To scan for a specific literal string, which does not exceed 30 characters, contained in a document title the S command is provided. Command KS is a combination of the K command and S command. The K command is performed first and is followed by a S command if the item is not located. Commands using the scan function (KS and S) should only be used for special circumstances since retrieval time is increased (see User's Guide for examples of input commands).

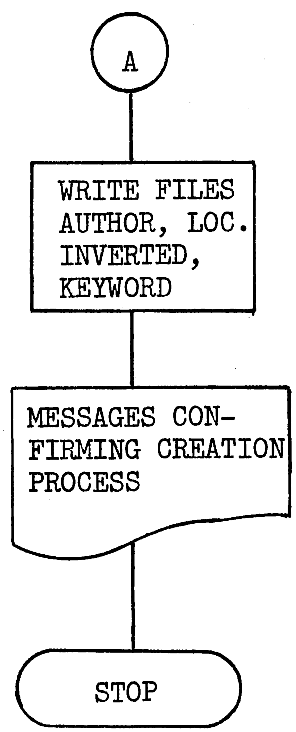
Interrogation is a simple process of determining the entry point into the data base and selecting the information. The entry is selected from the command specified and a search through the corresponding entry point AVL tree is performed. If one of the scan functions is requested, then a sequential search of the entire document file is performed printing the information requested if the stem is found.

APPENDIX E

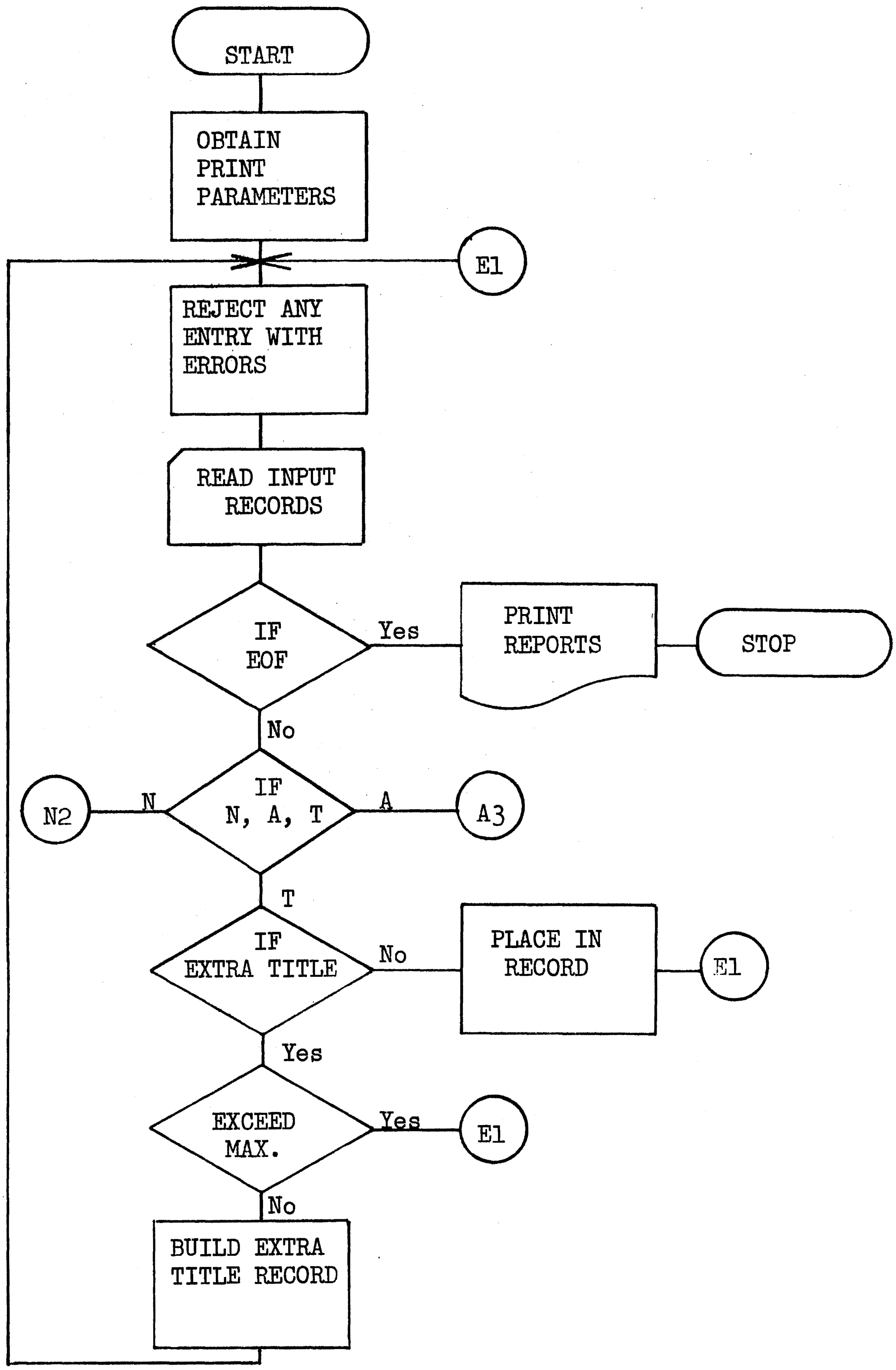
PROGRAM FLOWCHART

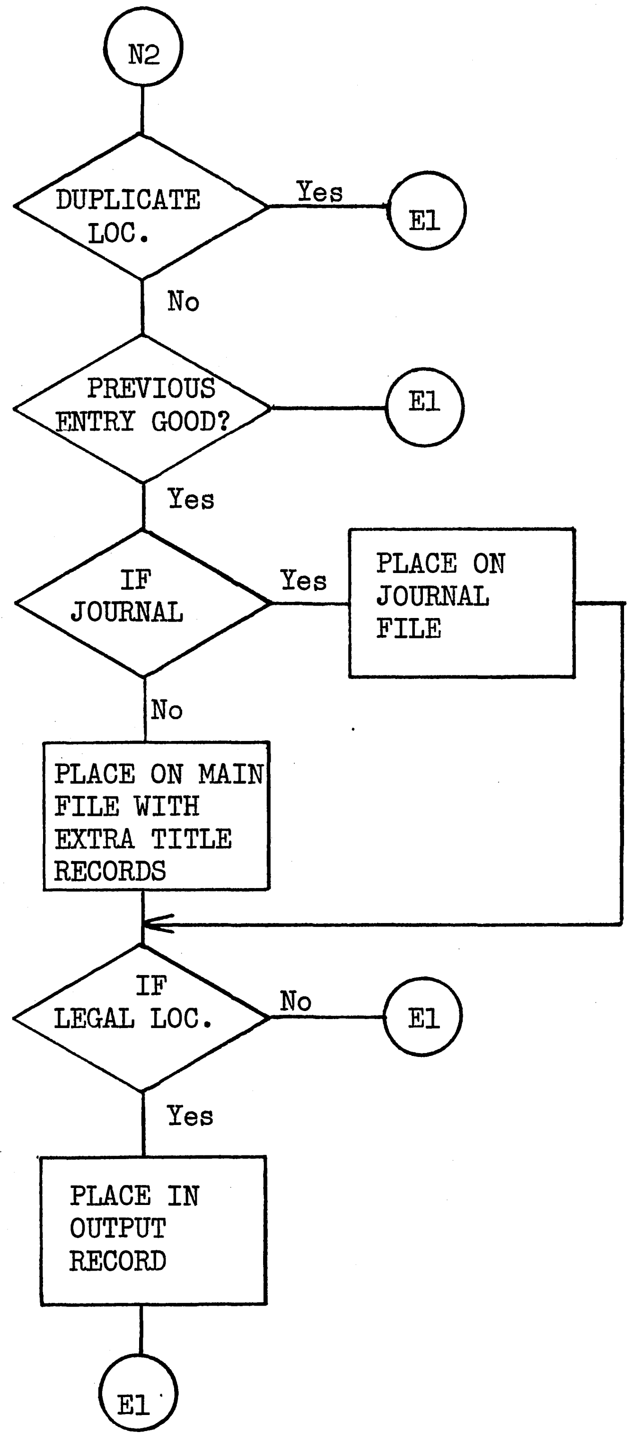
Creation

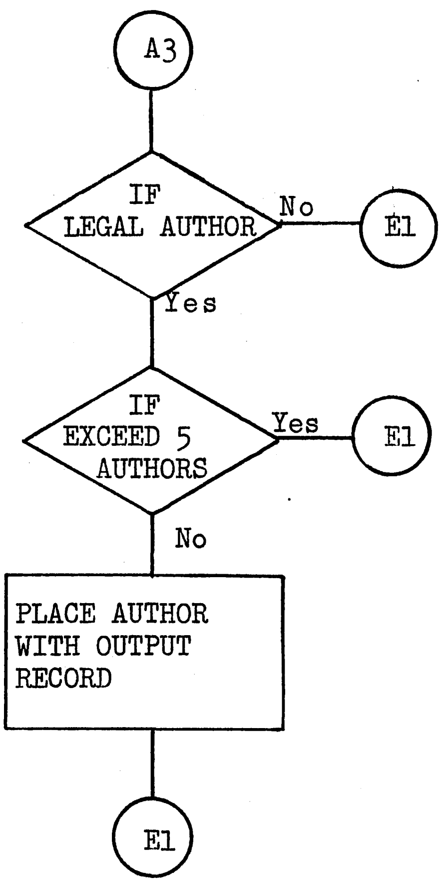

START
OPEN FILES
AUTHOR (SEQ.)
LOCATION (SEQ.)
KEYWORD (SEQ.)
OPEN FILES
INVERTED (REG)
DOCUMENT (REG)
LINK RLINKS OF
AUTHOR, LOCATION
AND KEYWORD FILE
LINK FOURTH
LINK OF
INVERTED FILE
PLACE "JOURNAL"
AND "THESIS" IN
KEYWORD FILE
SET VALUES OF
COUNT FIELDS
IN RECORD 0 OF
FILES
A


A
WRITE FILES AUTHOR, LOC. INVERTED, KEYWORD
MESSAGES CON- FIRMING CREATION PROCESS
STOP

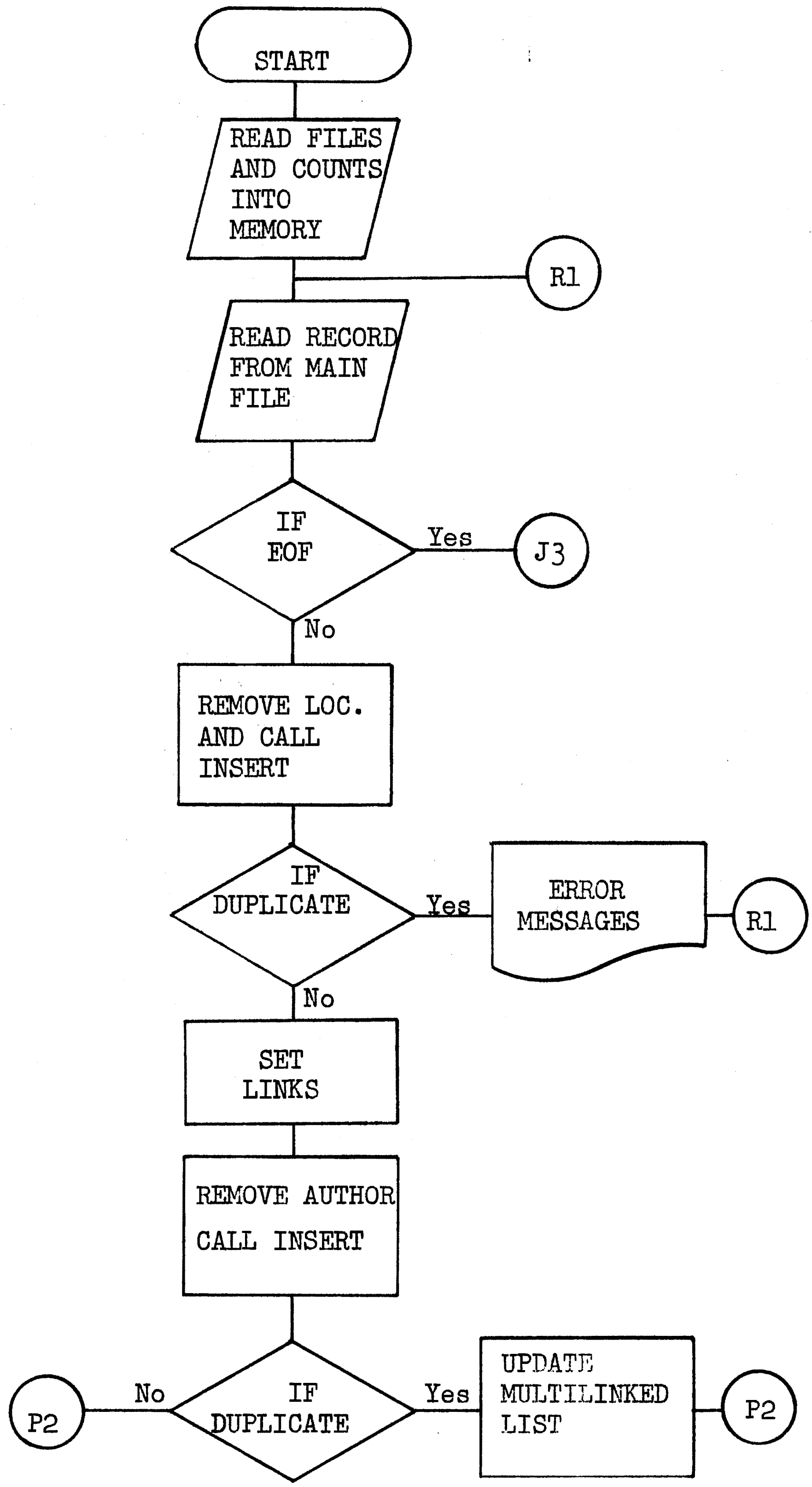
Edit

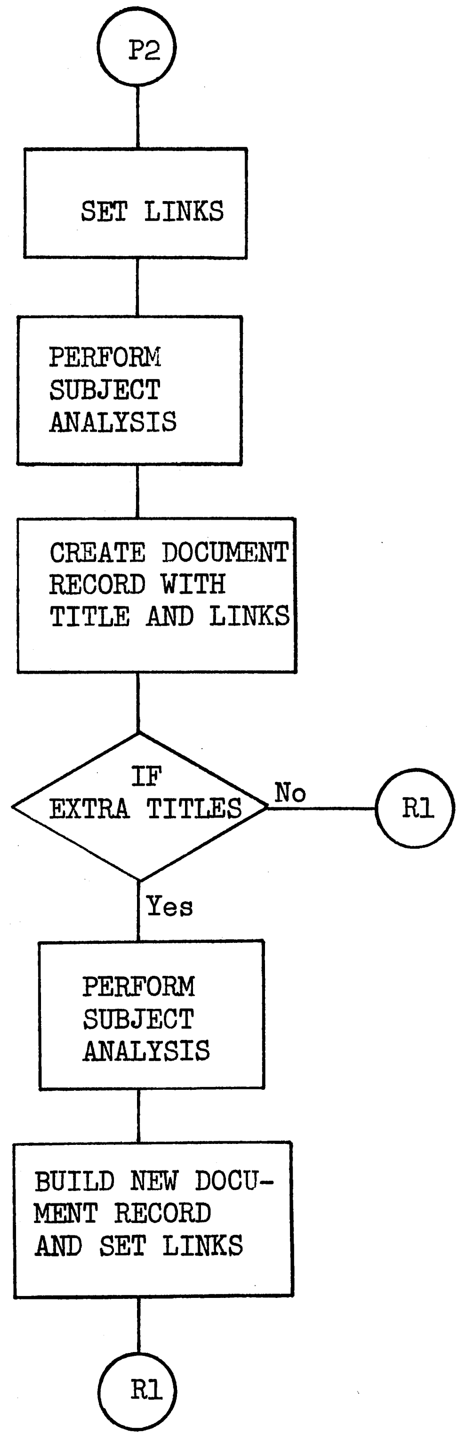

START
OBTAIN PRINT PARAMETERS
E1
REJECT ANY ENTRY WITH ERRORS
READ INPUT RECORDS
IF EOF
Yes
PRINT REPORTS
STOP
No
IF N, A, T
N
N2
A
A3
T
IF EXTRA TITLE
No
PLACE IN RECORD
E1
Yes
EXCEED MAX.
Yes
E1
No
BUILD EXTRA TITLE RECORD

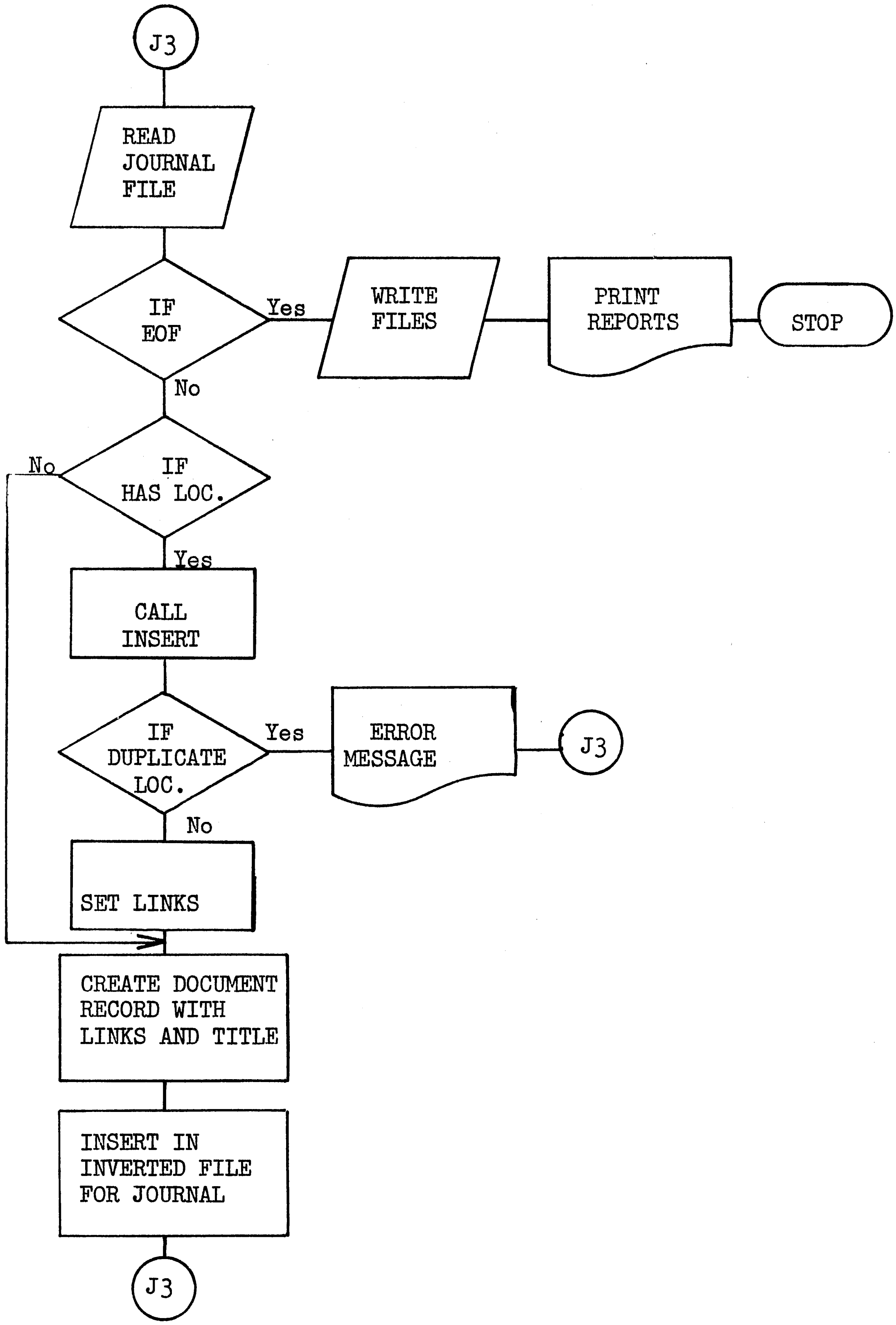

N2
DUPLICATE LOC.
Yes
E1
No
PREVIOUS ENTRY GOOD?
E1
Yes
IF JOURNAL
Yes
PLACE ON JOURNAL FILE
No
PLACE ON MAIN FILE WITH EXTRA TITLE RECORDS
IF LEGAL LOC.
No
E1
Yes
PLACE IN OUTPUT RECORD
E1


A3
IF
LEGAL AUTHOR
No
E1
Yes
IF
EXCEED 5
AUTHORS
Yes
E1
No
PLACE AUTHOR
WITH OUTPUT
RECORD
E1

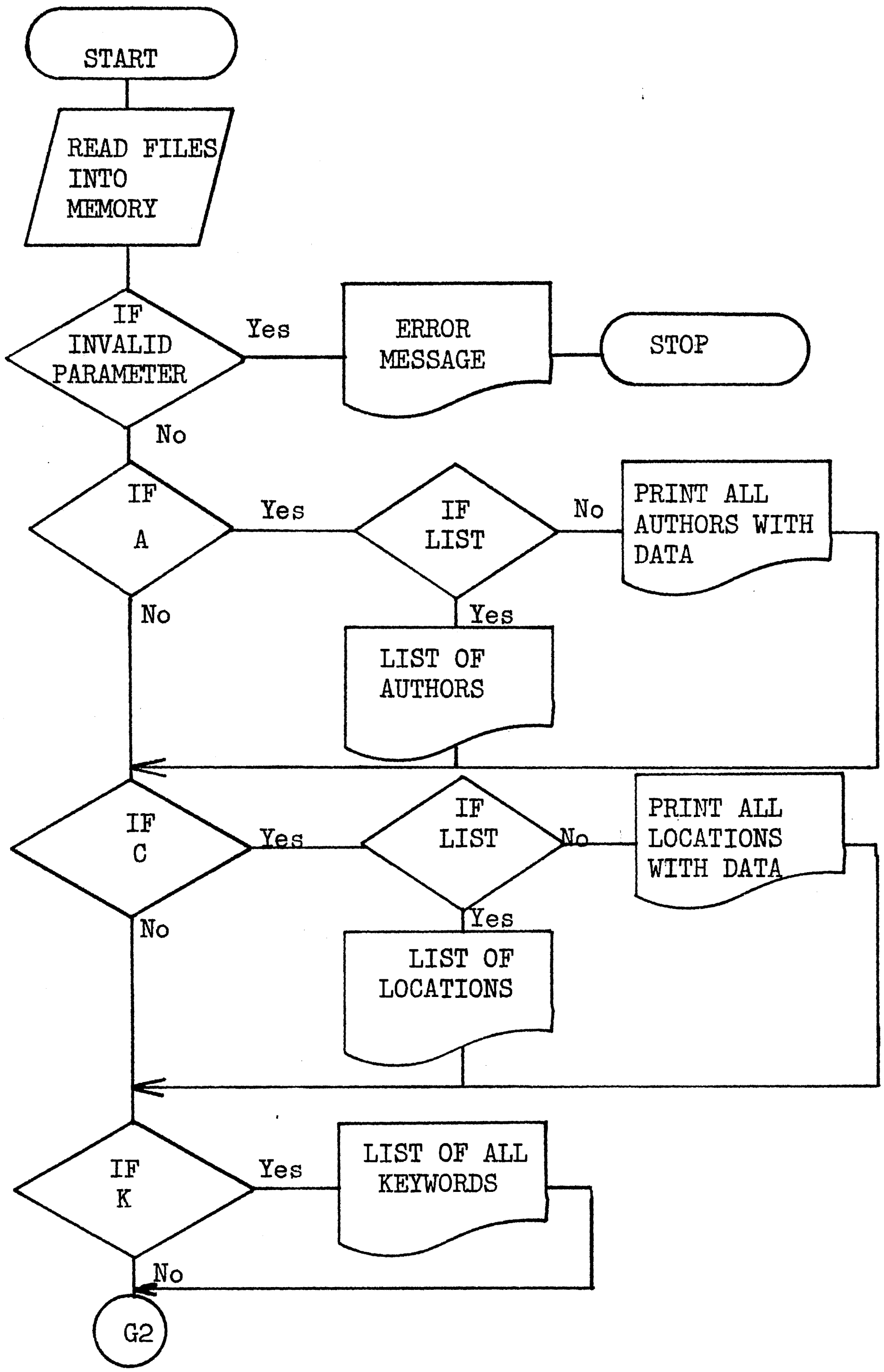
Update

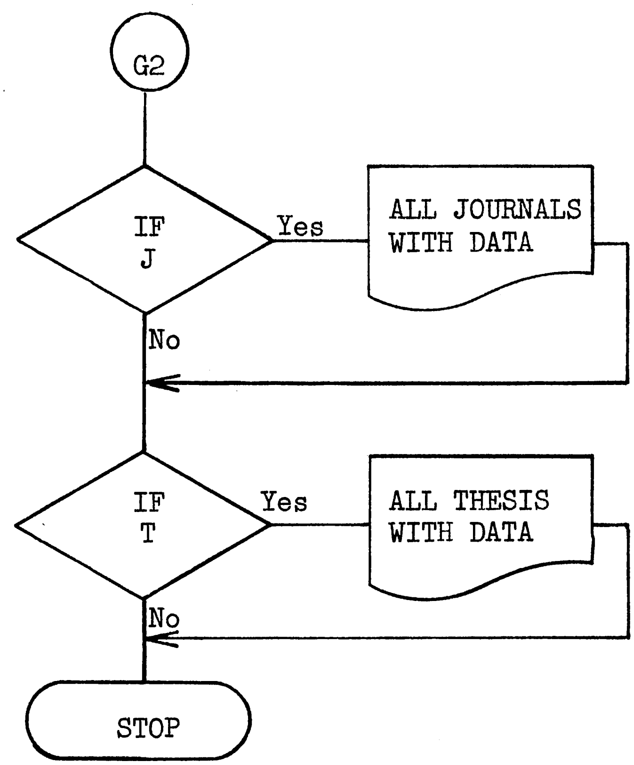

START
READ FILES AND COUNTS INTO MEMORY
R1
READ RECORD FROM MAIN FILE
IF EOF
Yes
J3
No
REMOVE LOC. AND CALL INSERT
IF DUPLICATE
Yes
ERROR MESSAGES
R1
No
SET LINKS
REMOVE AUTHOR CALL INSERT
P2
No
IF DUPLICATE
Yes
UPDATE MULTILINKED LIST
P2


P2
SET LINKS
PERFORM SUBJECT ANALYSIS
CREATE DOCUMENT RECORD WITH TITLE AND LINKS
IF EXTRA TITLES
No
R1
Yes
PERFORM SUBJECT ANALYSIS
BUILD NEW DOCU- MENT RECORD AND SET LINKS
R1

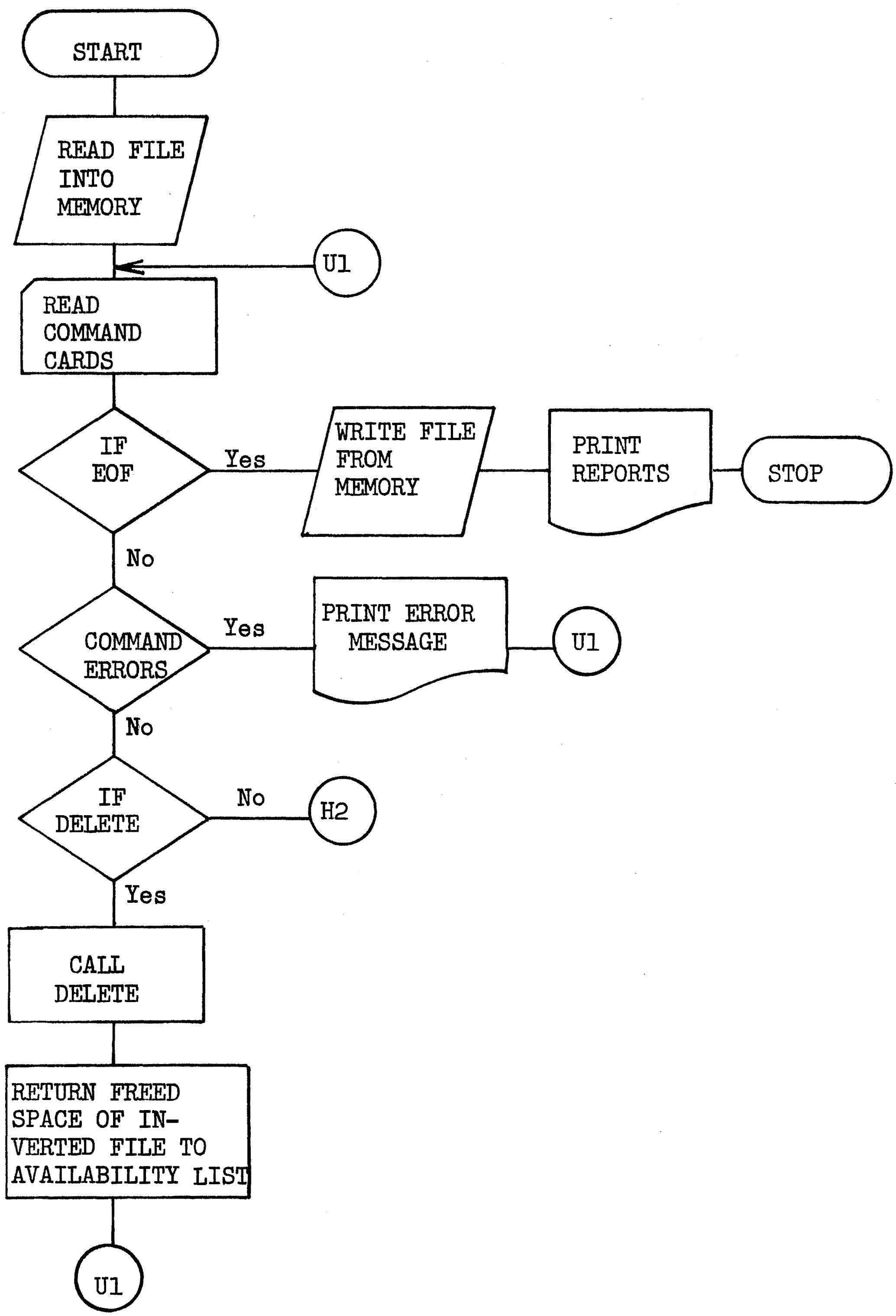

J3
READ
JOURNAL
FILE
IF
EOF
Yes
WRITE
FILES
PRINT
REPORTS
STOP
No
IF
HAS LOC.
No
Yes
CALL
INSERT
IF
DUPLICATE
LOC.
Yes
ERROR
MESSAGE
J3
No
SET LINKS
CREATE DOCUMENT
RECORD WITH
LINKS AND TITLE
INSERT IN
INVERTED FILE
FOR JOURNAL
J3

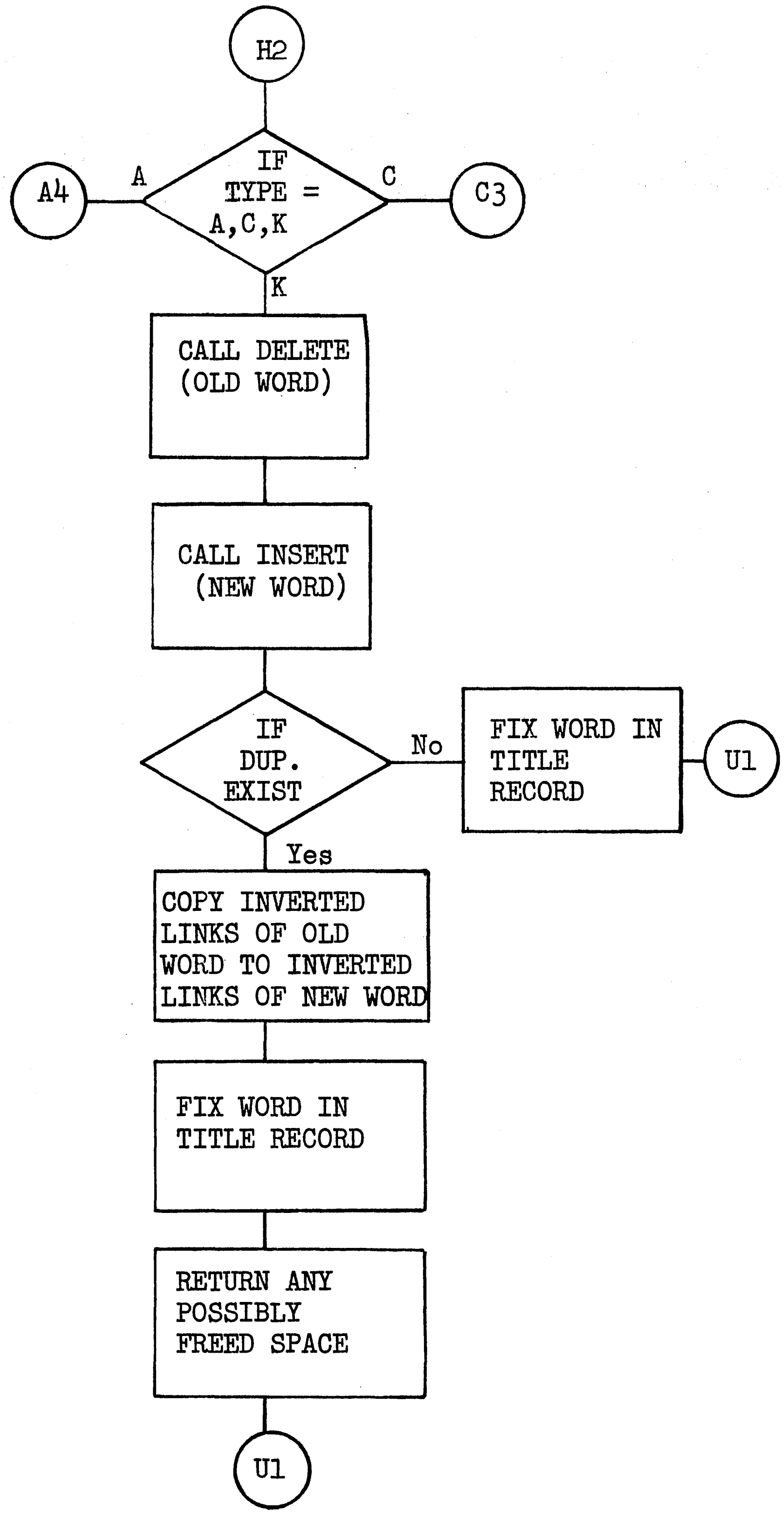
Print

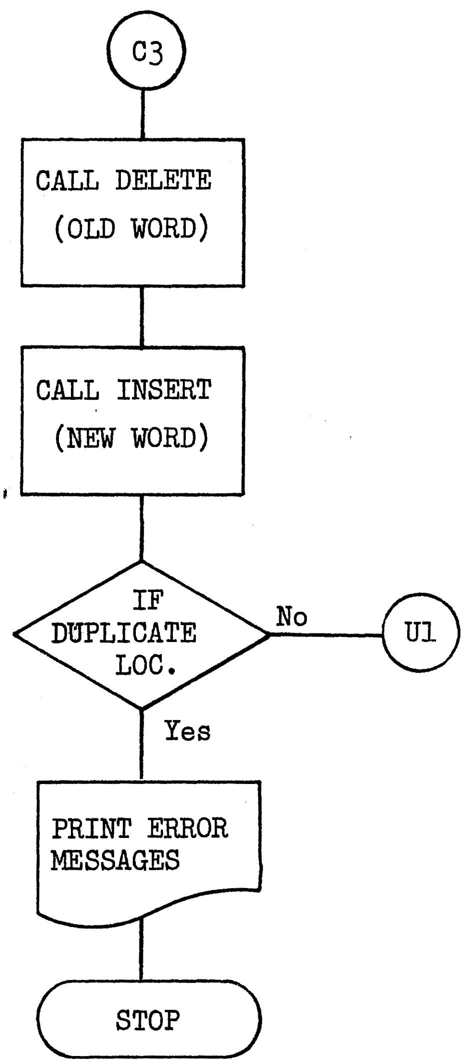

START
READ FILES INTO MEMORY
IF INVALID PARAMETER
Yes
ERROR MESSAGE
STOP
No
IF A
Yes
IF LIST
No
PRINT ALL AUTHORS WITH DATA
Yes
LIST OF AUTHORS
No
IF C
Yes
IF LIST
No
PRINT ALL LOCATIONS WITH DATA
Yes
LIST OF LOCATIONS
No
IF K
Yes
LIST OF ALL KEYWORDS
No
G2

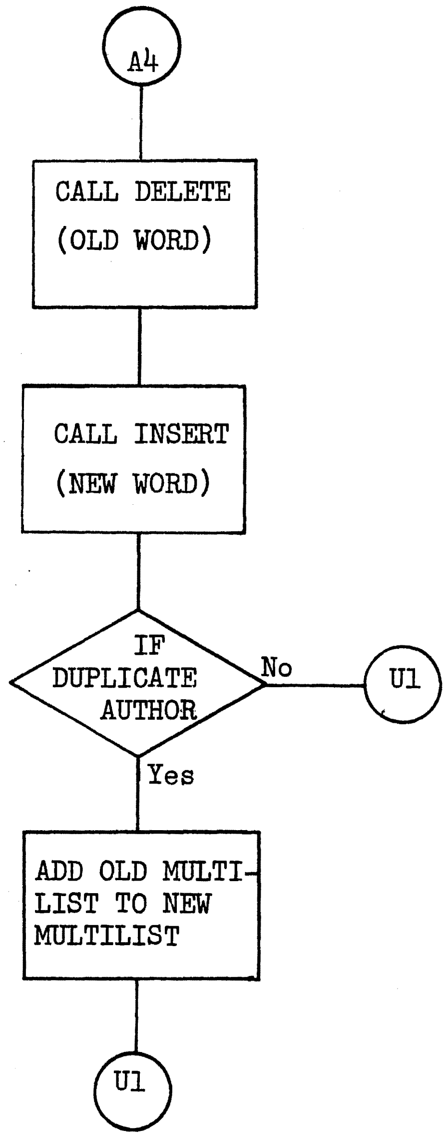

G2
IF
J
Yes
ALL JOURNALS
WITH DATA
No
IF
T
Yes
ALL THESIS
WITH DATA
No
STOP

Utility

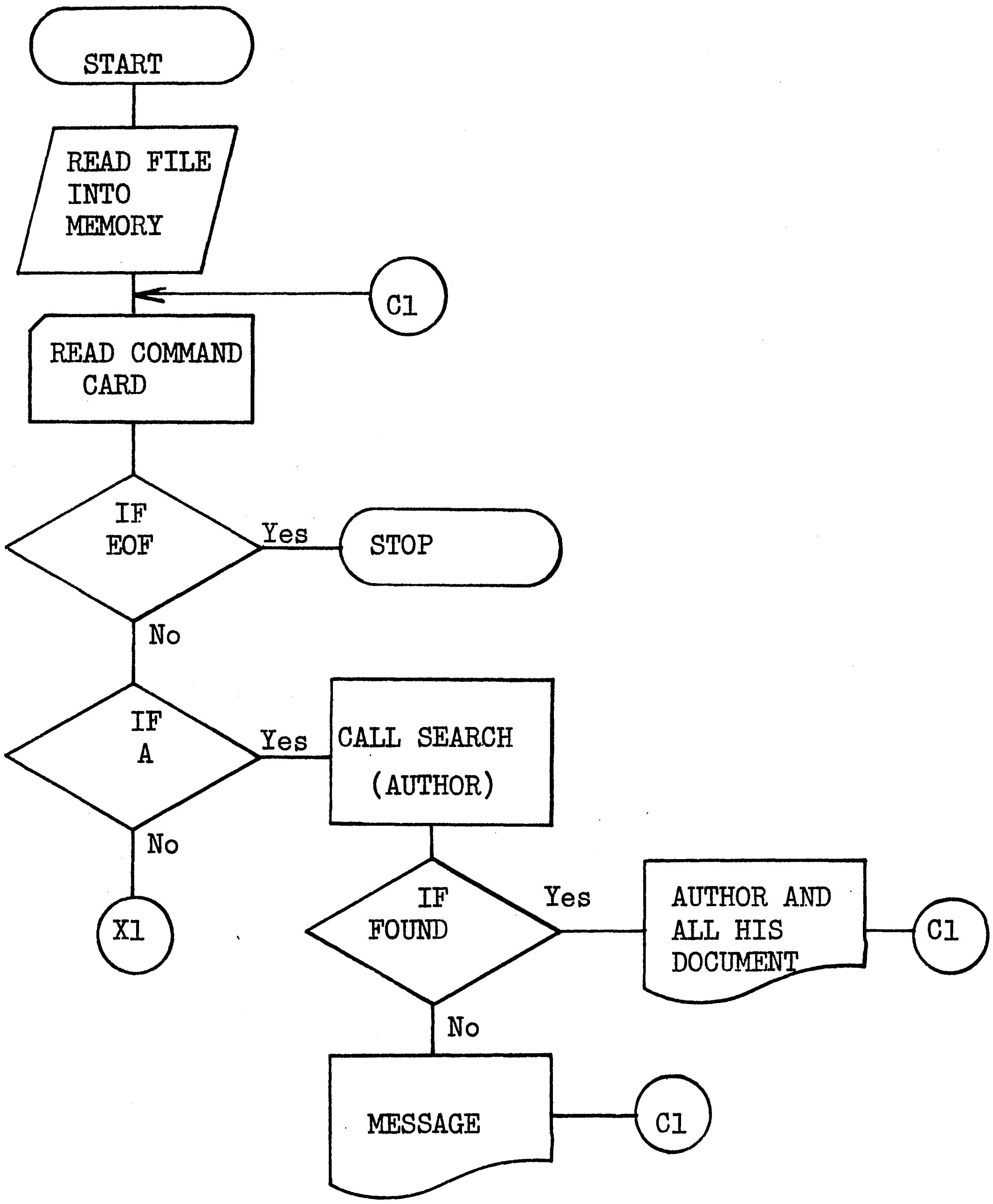

START
READ FILE INTO MEMORY
U1
READ COMMAND CARDS
IF EOF
Yes
WRITE FILE FROM MEMORY
PRINT REPORTS
STOP
No
COMMAND ERRORS
Yes
PRINT ERROR MESSAGE
U1
No
IF DELETE
No
H2
Yes
CALL DELETE
RETURN FREED SPACE OF IN-VERTED FILE TO AVAILABILITY LIST
U1

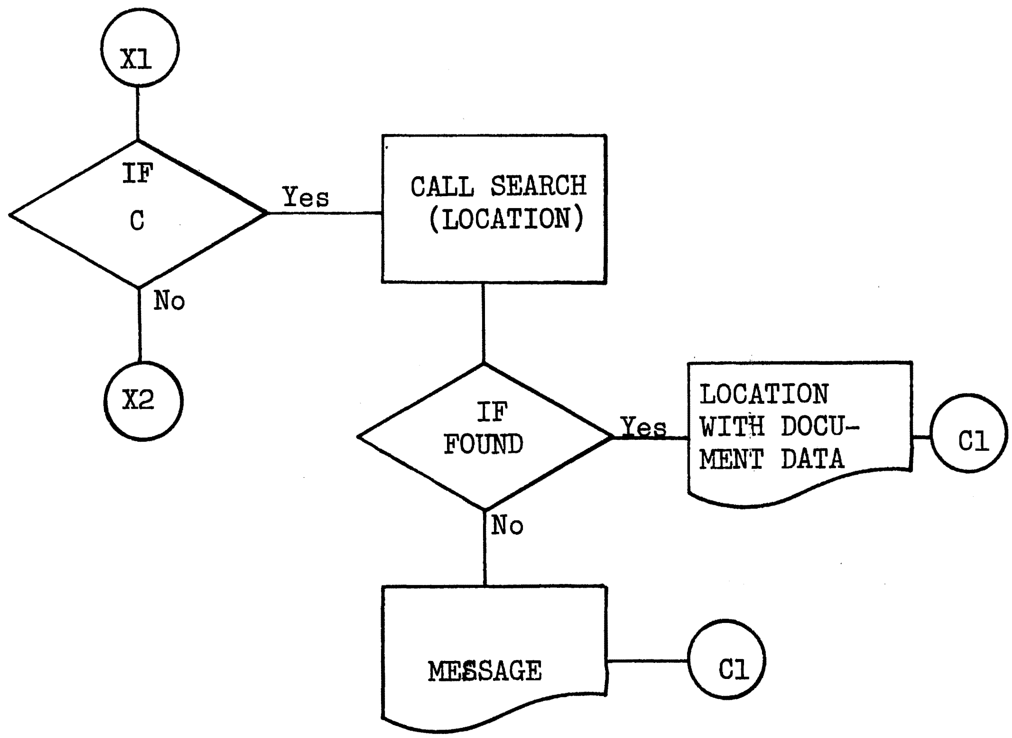

H2
A4
A
IF
TYPE =
A,C,K
C
C3
K
CALL DELETE
(OLD WORD)
CALL INSERT
(NEW WORD)
IF
DUP.
EXIST
No
FIX WORD IN
TITLE
RECORD
U1
Yes
COPY INVERTED
LINKS OF OLD
WORD TO INVERTED
LINKS OF NEW WORD
FIX WORD IN
TITLE RECORD
RETURN ANY
POSSIBLY
FREED SPACE
U1

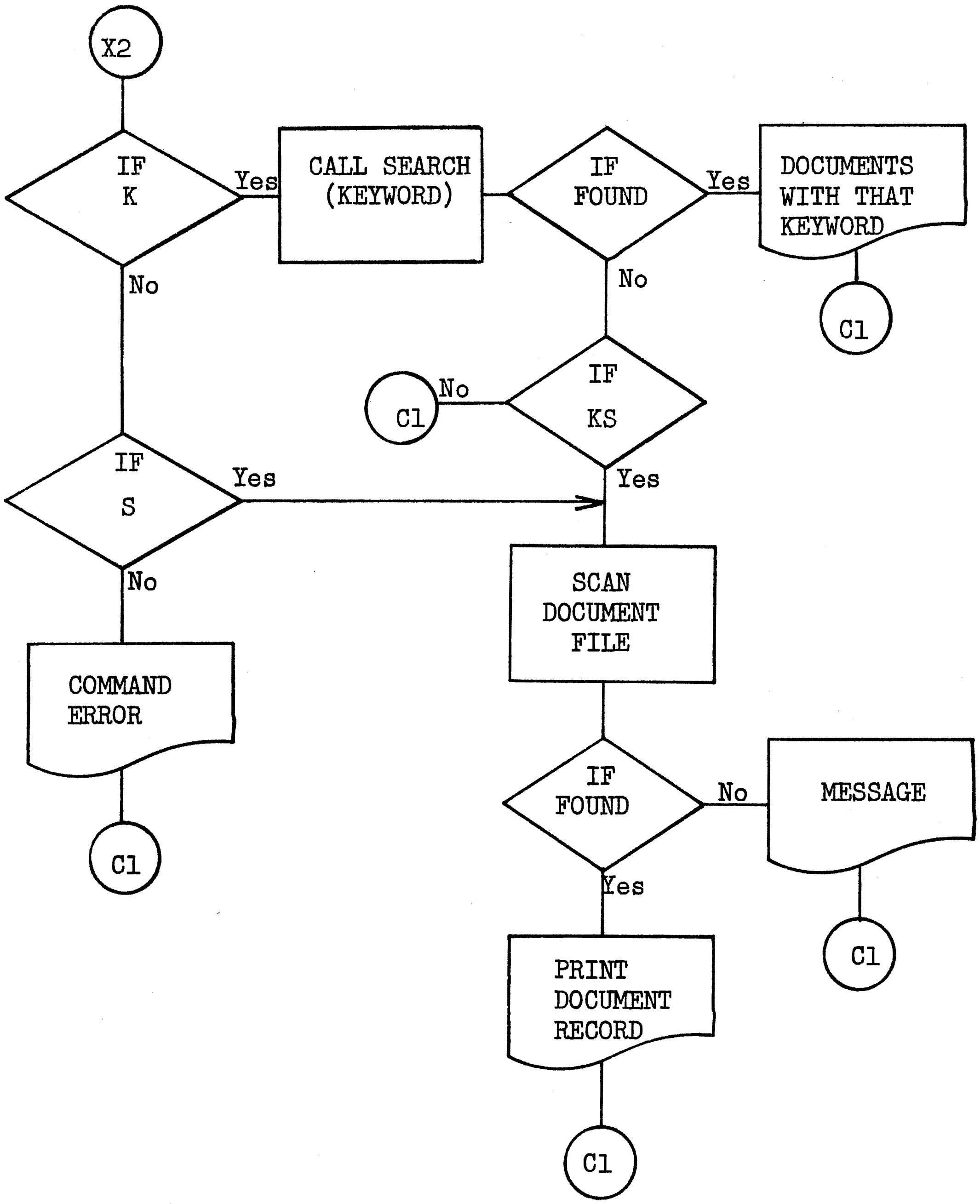

C3
CALL DELETE
(OLD WORD)
CALL INSERT
(NEW WORD)
IF
DUPLICATE
LOC.
No
U1
Yes
PRINT ERROR
MESSAGES
STOP


A4
CALL DELETE
(OLD WORD)
CALL INSERT
(NEW WORD)
IF
DUPLICATE
AUTHOR
No
U1
Yes
ADD OLD MULTI-
LIST TO NEW
MULTILIST
U1

Interrogation




X1
IF
C
Yes
No
X2
CALL SEARCH
(LOCATION)
IF
FOUND
Yes
No
LOCATION
WITH DOCU-
MENT DATA
C1
MESSAGE
C1



X2
IF K
Yes
CALL SEARCH (KEYWORD)
IF FOUND
Yes
DOCUMENTS WITH THAT KEYWORD
C1
No
No
IF KS
No
C1
IF S
Yes
Yes
No
SCAN DOCUMENT FILE
COMMAND ERROR
C1
IF FOUND
No
MESSAGE
Yes
C1
PRINT DOCUMENT RECORD
C1

APPENDIX F

PROGRAM OUTPUT

CREATION OF THE BIBLIOGRAPHIC DATA BASE

THE FOLLOWING FILES HAVE BEEN CREATED: AUTHOR FILE,LOCATION FILE,KEYWORD FILE,INVERTED FILE,AND TITLE FILE.
*** THE TWO LOGICAL FILES FOR JOURNALS AND THESIS HAVE BEEN ESTABLISHED ***

MAX. NO. OF AUTHORS	MAX. NO. OF LOCATIONS	MAX. NO. OF KEYWORDS
2500	2000	1000

MAX. NO. OF INVERTED RECORDS	MAX. NO. OF TITLE RECORDS
2000	2000

NORMAL END OF JOB

THE FOLLOWING ENTRIES WERE ADDED

629.8920151 5795AE 00614
STARKE P
ABSTRACT AUTOMATA.

001.642 C7410 00615
SAYERS A COMTRE C
OPERATING SYSTEMS SURVEY.

511.6 C731 1970 00616
GENERAL A
PROCEEDINGS OF THE SECOND CHAPEL HILL CONFERENCE ON COMBINATORIAL MATH
EMATICS AND ITS APPLICATIONS.

658.4002854 H432M 00617
HEAD R
MANAGER'S GUIDE TC MANAGEMENT INFORMATION SYSTEMS.

658.403 I43 00618
GRUENBERGER F
INFORMATION SYSTEMS FOR MANAGEMENT.

029.7 K52E 00619
KING D BRYANT E
THE EVALUATION OF INFORMATION SERVICES AND PRODUCTS.

658.8CC285 C639D 00620
CLIFTON H
CATA PROCESSING SYSTEMS DESIGN.

658.054 S998I 00621
SZWEDA R
INFORMATION PROCESSING MANAGEMENT.

001.6442 S989C 00622
GENERAL A
IEEE CONFERENCE RECORD OF THE SYMPOSIUM ON FEATURE EXTRACTION AND SELE
CTION IN PATTERN RECOGNITION.

001.6424 A396S 1973 00623
GENERAL A
SAN FRANCISCO CONFERENCE ON ALGOL 68 IMPLEMENTATION, 1973 (PROCEEDINGS

**** THE FOLLOWING ENTRIES WERE NOT ADDED DUE TO ERRORS LISTED ****

DIAGONSTICS	ERROR CARD	REJECTED ENTRY
**** BLANK MISSING IN CALL NUMBER	00639010	00639
**** BLANK MISSING IN CALL NUMBER	00650010	00650
**** MISSING CALL NUMBER FOR THIS ENTRY	00700000	B/
**** TITLE CARD MISSING FOR	00720	00720
**** 72 DOES NOT CONTAIN N,A,OR T	00800000	00800
**** TITLE EXCEEDEDS 420 CHAR EXCEEDING CARD IS	00810000	00810
**** MISSING CALL NUMBER FOR THIS ENTRY	-0860000	A I $
**** TITLE CARD MISSING FOR	00860	00860

LIST OF ALL JOURNALS

370.1805 A105
AEDS JOURNAL.

501 C9935
CYBERNETICS.(JOURNAL).

010.5 A512
JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE.

ACTA INFORMATICA.(JOURNAL).

CIPS-COMPUTER MAGAZINE.(JOURNAL).

COMPUTER DECISIONS.(JOURNAL).

INTERNATIONAL JOURNAL OF COMPUTER AND INFORMATION SCIENCES.

HONEYWELL COMPUTER JOURNAL.

TOTAL NUMBER OF JOURNALS = 8

LIST OF NEW AUTHORS ADDED

STARKE P	SAYERS A	COMTRE C	GENERAL A
HEAD R	GRUENBERGER F	KING D	BRYANT E
CLIFTON H	SZWEDA R	PRAGER W	WINOGRAD T
LONDON K	WAITE W	GOLDSTONE H	KARBOWIAK A
HUEY R	MINKER J	ROSENFELD S	PETRICK S
FINDLER N	PFALTZ J	BERNSTEIN H	RUSTIN R
PETERS S	JOHNSON R	KAST F	ROSENWEIG J
PRICE W	FREIBERGER W	GRENADER J	MARGOLIN B
TSAO R	DONALD A	COLIN A	CHARTRAND R
WELLS M	WALKER D	PETROCELLI O	SIMON H
SIKLOSSY L	ELLIOT R	BRENDER R	READ R
CLARE C	GAVRILOV M	ZAKREVSKII A	MADLER M
ORGANICK E	MAURER W	HAMBLEN J	LAW E
U. S	PAINTER J	MCGOWAN C	GOOD D
WOOLLONS D	DONOVAN J	HELLERMAN H	BRENT R
WILKS Y	SOUCEK B	KORN G	FLORES I
STOUTEMYER D	HEAPS H	HETZEL W	REITMAN J
METZGER R	LUCAS H	KEYS W	CASHMAN T
HOLT A	CHAGNON S	SHAPIRO R	MARSHALL S

TOTAL NUMBER OF AUTHORS ADDED = 76

LIST OF NEW CALL NUMBERS ADDED

629.8920151 5795AE	001.642 C7410	511.6 C731 1970	658.4002854 H432M
658.403 I43	029.7 K52E	658.800285 C639D	658.054 S998I
001.6442 S989C	001.6424 A396S 1973	001.6424 A1105P8	420.285 W776U
001.6 L847D	001.6425 W145I	621.3819509 G624C	003 K18I
010.78 S988 1971	512.020285 S989 1971	001.6424 F744F4	001.6425 C858D
001.6424 C858F	410.6 G573	658.4032 J68T 1973	658.4032 P946G
001.64 S797	658 D675M	001.6425 C696I	301.243 C7385
511.602854 W455E	029.7 I61	001.6408 P497B	001.535 S594P
001.6423 E46P	001.64404 C858C 1970	510.7805 B105	574.018 B837P 1973
511.5 R284G	621.3819582 C591D	001.6424 L111	001.6406 C738C
001.64 068M	001.642 M453P 1972	338.47 C73 H199C	EIU97 D598 DEC 1972
001.6425 P148S	001.6425 M146C	001.642 G46T	621.3819 C73 W919I
001.642 C858D	001.642 D687S	001.64044 H477D 1973	515.62 B839A
410 W688G	001.64044 S719M	001.64044 K84M	001.642 F634J
001.6424 P110158	001.6424 H434I	001.6406 A938 1969V1	001.6406 A938 1969 V2
001.6425 P964	001.424 R379C	001.642 M596M	658.403 L933C
001.64 K44B	EDE16 A111 AD626819	370.1805 A105	501 C9935
010.5 A512			

TOTAL NUMBER OF LOCATIONS ADDED = 69

SUMMARY OF TOTALS

NO. OF AUTHOR	NO. OF CALL NUMBERS	NO. OF KEYWORDS
76	69	92

NO. OF TITLE RECORDS	NO. OF KEYWORD RECORDS	NO. OF UPDATE ERRORS
74	95	1

NORMAL END OF PROGRAM

LIST OF NEW KEYWORDS ADDED

AUTOMATA	OPERATING	SURVEY	CHAPEL	HILL
COMBINATORIAL	MANAGER	GUIDE	EVALUATION	SERVICES
PRODUCTS	IEEE	RECORD	EXTRACTION	SELECTION
PATTERN	RECOGNITION	IMPLEMENTATION	UNDERSTANDING	NATURAL
DECISION	TABLES	SOFTWARE	PASCAL	NEUMANN
STORAGE	RETRIEVAL	SYMBOLIC	MANIPULATION	HIGH
EXTENSIONS	FORTRAN	SLIP	AMPPL	TREETRAN
OPTIMIZATION	COURANT	FORMAL	SEMANTICS	GOALS
LINGUISTIC	GRAPHS	NETWORKS	PERFORMANCE	ELEMENTS
INTERACTIVE	BIBLIOGRAPHIC	SEARCH	PAPERS	REPRESENTATION
MEANING	EXPERIMENTS	SOLVING	FLOWCHARTING	CELLULAR
SPACES	LOGIC	USING	SOLID	STATE
LYAPAS	CODING	MULTICS	EXAMINATION	STRUCTURE
MANPOWER	SUPPLY	DEMAND	DIRECTORY	CRIMINAL
JUSTICE	CORRECTNESS	RESULTS	LAMBDA	CALCULUS
PROVING	BASE	MINIMIZATION	DERIVATIVES	GRAMMAR
MINICOMPUTERS	ENGINEERS	FILE	DEFINITION	ADELAIDE
TEST	DISCRETE	SYNTHESIS	COMPLEX	ORGANIZATIONS

TOTAL NUMBER OF KEYWORDS ADDED = 90

LIST OF ALL ITEMS WITH CALL NUMBERS

EDE16 A111 AD626819
HOLT A   CHAGNON S   SHAPIRO R   MARSHALL S
INFORMATION SYSTEM THEORY PROJECT, VOLUME 1: G-THEORY.

EIU97 C598 DEC 1972
LAW E   U. S
1972 DIRECTORY OF AUTOMATED CRIMINAL JUSTICE INFORMATION SYSTEMS.

001.424 R379C
REITMAN J
COMPUTER SIMULATION APPLICATIONS: DISCRETE-EVENT SIMULATION FOR SYNTHE
SIS AND ANALYSIS OF COMPLEX SYSTEMS.

001.535 S594F
SIMON H   SIKLOSSY L
REPRESENTATION AND MEANING: EXPERIMENTS WITH INFORMATION PROCESSING SY
STEMS.

001.6 L847D
LONDON K
DECISION TABLES.

001.64 K44B
KEYS W   CASHMAN T
BASIC PRINCIPLES OF DATA PROCESSING.

001.64 S797
FREIBERGER W   GRENADER J   MARGOLIN B   TSAO R
STATISTICAL COMPUTER PERFORMANCE EVALUATION.

001.64 068M
ORGANICK E
THE MULTICS SYSTEM: AN EXAMINATION OF ITS STRUCTURE.

001.64044 H477D 1973
HELLERMAN H
DIGITAL COMPUTER SYSTEM PRINCIPLES (2-ND ED.).

001.64044 K84M
KORN G
MINICOMPUTERS FOR ENGINEERS AND SCIENTISTS.

LIST OF ALL AUTHORS WITH THEIR PUBLICATIONS

BERNSTEIN H
001.6424 F744F4
FINDLER N PFALTZ J
FOUR HIGH-LEVEL EXTENSIONS TO FORTRAN IV: SLIP, AMPPL-II, TREETRAN, SY
MBOLANG.

BRENDER R
574.018 B837P 1973

A PROGRAMMING SYSTEM FOR THE SIMULATION OF CELLULAR SPACES (THESIS.).

BRENT R
515.62 B839A

ALGORITHMS FOR MINIMIZATION WITHOUT DERIVATIVES.

BRYANT E
029.7 K52E
KING D
THE EVALUATION OF INFORMATION SERVICES AND PRODUCTS.

CASHMAN T
001.64 K44B
KEYS W
BASIC PRINCIPLES OF DATA PROCESSING.

CHAGNON S
EDE16 A111 AD626819
HOLT A SHAPIRO R MARSHALL S
INFORMATION SYSTEM THEORY PROJECT, VOLUME 1: G-THEORY.

CHARTRAND R
301.243 C7385

COMPUTERS IN THE SERVICE OF SOCIETY.

CLARE C
621.3819582 C591D

DESIGNING LOGIC SYSTEMS USING SOLID STATE MACHINES.

****THE FCLLOWING ENTRIES ARE JOURNALS

370.1805 A105

AEDS JOURNAL. 000651

501 C9935

CYBERNETICS.(JOURNAL). 000656

010.5 A512

JOUFNAL CF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE. 000657

ACTA INFORMATICA.(JOURNAL). 000652

CIPS-COMPUTER MAGAZINE.(JOURNAL). 000654

COMPUTER DECISIONS.(JOURNAL). 000655

INTEPNATIONAL JOURNAL OF COMPUTER AND INFORMATION SCIENCES. 000658

HCNEYWELL COMPUTER JCURNAL. 000659

LIST OF ALL KEYWORDS

ADELAIDE	AMPPL	AUTOMATA	
BASE	BIBLIOGRAPHIC		
CALCULUS	CELLULAR	CHAPEL	CODING
COMBINATORIAL	COMPLEX	CORRECTNESS	COURANT
CRIMINAL			
DECISION	DEFINITION	DEMAND	DERIVATIVES
DIRECTORY	DISCRETE		
ELEMENTS	ENGINEERS	EVALUATION	EXAMINATION
EXPERIMENTS	EXTENSIONS	EXTRACTION	
FILE	FLOWCHARTING	FORMAL	FORTRAN
GOALS	GRAMMAR	GRAPHS	GUIDE
HIGH	HILL		
IEEE	IMPLEMENTATION	INTERACTIVE	
JOURNAL	JUSTICE		
LAMBDA	LINGUISTIC	LOGIC	LYAPAS
MANAGER	MANIPULATION	MANPOWER	MEANING
MINICOMPUTERS	MINIMIZATION	MULTICS	
NATURAL	NETWORKS	NEUMANN	
OPERATING	OPTIMIZATION	ORGANIZATIONS	
PAPERS	PASCAL	PATTERN	PERFORMANCE
PRODUCTS	PROVING		
RECOGNITION	RECORD	REPRESENTATION	RESULTS
RETRIEVAL			
SEARCH	SELECTION	SEMANTICS	SERVICES
SLIP	SOFTWARE	SOLID	SOLVING
SPACES	STATE	STORAGE	STRUCTURE
SUPPLY	SURVEY	SYMBOLIC	SYNTHESIS
TABLES	TEST	THESIS	TREETRAN
UNDERSTANDING	USING		

LIST OF ALL AUTHORS

INTERROGATION OF THE BIBLIOGRAPHIC DATA BASE

THE FOLLOWING LOCATION WAS REQUESTED: 574.018 B837P 1973

574.018 B837P 1973
BRENDER R
A PROGRAMMING SYSTEM FOR THE SIMULATION OF CELLULAR SPACES (THESIS.).

THE FOLLOWING AUTHOR WAS REQUESTED: REITMAN J

REITMAN J
001.424 R379C

COMPUTER SIMULATION APPLICATIONS: DISCRETE-EVENT SIMULATION FOR SYNTHE
SIS AND ANALYSIS OF COMPLEX SYSTEMS.

THE FOLLOWING KEYWORD WAS REQUESTED: LOGIC

621.3819582 C591D
CLARE C
DESIGNING LOGIC SYSTEMS USING SOLID STATE MACHINES.

001.6424 L111
GAVRILOV M ZAKREVSKII A MADLER M
LYAPAS: A PROGRAMMING LANGUAGE FOR LOGIC AND CODING ALGORITHMS.

VITA

Perry Lee Ball

Candidate for the Degree of

Master of Science

Thesis: THE DESIGN OF A BIBLIOGRAPHIC DATA BASE SYSTEM

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Baton Rouge, Louisiana, May 21, 1951, the son of Mr. and Mrs. T. W. Ball.

Education: Graduated from Redemptorist High School, Baton Rouge, Louisiana, in May, 1969; received Bachelor of Science Degree from Northwestern State University of Louisiana, Natchitoches, Louisiana, in May, 1973, with a major in Mathematics and with a minor in Computer Science; completed requirements for the Master of Science degree at Oklahoma State University, Stillwater, Oklahoma, in May, 1975.

Professional Experience: Programmer, Blue Cross of Louisiana, Baton Rouge, Louisiana, May, 1973, to December, 1973; Graduate Assistant, Oklahoma State University, Computing and Information Sciences Department, Stillwater, Oklahoma, January, 1974, to May, 1975.