DOCUMENT RESUME

ED 057 806 LI 003 295

AUTHOR de Boer, Aeint
TITLE Center for Information Services, Phase II: Detailed System Design and Programming, Part 1 - A Modular Computer Program for Reference Retrieval, Phase IIA Final Report.
INSTITUTION California Univ., Los Angeles. Inst. of Library Research.
SPONS AGENCY National Science Foundation, Washington, D.C.
PUB DATE 1 Mar 71
NOTE 45p.; (21 References)

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS *Cataloging; *Computer Programs; *Information Centers; *Information Retrieval; *Information Services; Library Acquisition
IDENTIFIERS *Computer Software; University of California (Los Angeles)

ABSTRACT

A modular computer program for subject access to multiple fixed-vocabulary bibliographic data bases has been developed. It is written in PL/I and Assembler Language for operation on the IBM 360 Model 91 computer at the University of California, Los Angeles, and can run on a computer as small as an IBM 360 Model 50. The program (MULFSCH) described represents an intermediate stage in software development for the Center for Information Services (CIS). As such, its principal value to the CIS project is in development of the various procedures (acquisition, cataloging, retrieval, etc.) attendant to the operation of the CIS. (Related documents are available as LI 003296 through LI 003301). (Author)

CENTER FOR INFORMATION SERVICES
PHASE II: DETAILED SYSTEM DESIGN AND PROGRAMMING

NSF GRANT GN-827

PHASE IIA FINAL REPORT
PART 1

A MODULAR COMPUTER PROGRAM FOR REFERENCE RETRIEVAL

by

Aeint de Boer

1 March 1971

Institute of Library Research
University of California
Los Angeles, California

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

ABSTRACT

A modular computer program for subject access to multiple fixed-vocabulary bibliographic data bases has been developed. It is written in PL/I and Assembler Language for operation on the IBM 360 Model 91 computer at the University of California, Los Angeles, and can run on a computer as small as an IBM 360 Model 50. The program (MULFSCH) described represents an intermediate stage in software development for the Center for Information Services (CIS). As such, its principal value to the CIS project is in development of the various procedures (acquisition, cataloging, retrieval, etc.) attendant to the operation of the CIS.

I. BIBLIOGRAPHIC DATA BASES

GENERAL CHARACTERISTICS

The data bases available to the library, which include bibliographic as well as statistical or natural language, are produced by various federal agencies, professional societies and commercial organizations, and the number grows each year. Often produced in hard copy or recorded on film, these data bases also are available on magnetic tape due not only to the low cost per character and high capacity of the medium but to its use in facilitating computer based processing operations.

For the most part, bibliographic data bases are offered as by-products of on-going cataloging, abstracting and indexing services. As such, they were created primarily to support the on-going service and fit its requirements. A major consequence of this situation is that nearly every file is a unique case. Each file has its own format, data fields, coding schemes and processing requirements. Thus the predominant feature of bibliographic files is variability, both in the formats of individual files and in the composition of individual entries within a given file. The importance and advantages of standardization, at least in interchange formats (e.g. the MARC II format), are beginning to be realized, but it will be a while yet before the effects become widespread. In the data base discussions that follow:

a. record will refer to the information read from an external medium in a single machine operation (i.e. a physical record in common parlance)

b. entry will refer to the information for one citation

c. item will refer to a recognizable data element within an entry

d. group will refer to a set of related items

e. delimiter will refer to an item which marks the beginning or end of another item or group

f. tag will refer to an item whose value indicates the presence of another item or group.

PROBLEM AREAS

The entries for bibliographic files are universally of variable length. Normally the only prior knowledge that is available is the maximum length of an entry or record. An entry may be part of a single record, correspond exactly to a single record, or span a number of records. In any case, some means must be provided to allow the computer (program) to recognize a complete entry and to determine its length.

In some cases a file may contain a number of different entry or record types, each with its own format. Such files usually contain a special tag (entry or record type code) in a fixed location to indicate the nature of the current entry or record.

Often the presence of an item within a given entry is optional. For example, an entry may not have an author or a report number. This may be done by leaving the item blank, but the most common approach is to delete it entirely. In either case, the computer must be able to determine the presence or absence of an optional item.

The majority of the items in an entry are inherently of variable length. Although such items as author, title, and descriptor terms could be forced into standard fixed length fields, this would result in significant amounts of blank (wasted) space in an entry. In practice, this is rarely done. Thus, for most items, some means must be provided to allow the computer to determine the length of the item, and, since following items will change location relative to the beginning of an entry as the length of a preceding item changes, to determine the location of an item.

Some of the items in an entry may have multiple occurrences. The most common examples are multiple authorship and multiple subject access terms. Once again, space may be provided for a fixed maximum number of occurrences, but this is not common. If the number of occurrences is variable, the computer must be able to determine the actual number for any given entry.

Another problem, of a slightly different nature, is the character set used for a file. Current popular choices are BCD (Binary Coded Decimal), EBCDIC (Extended Binary Coded Decimal Interchange Code) and ASCII (American Standard Code for Information Interchange). A superior character set from the computer's viewpoint may, however, be inadequate for a particular application. For instance, for its MARC tapes, distributed on both 7-track and 9-track tapes, the Library of Congress uses an 8-bit extension of ASCII (a 7-bit code) with numerous additional diacritics and foreign letters on 9-track tapes. On 7-track tapes, they utilize a 6-bit code derived from ASCII with non-locking shift codes to indicate that the following character is from one of three

additional non-standard character sets. (16) In effect, this means that certain single print characters are written and stored as character pairs.

These then are some of the problems that must be faced by the system analyst when he is charged with designing a format for a bibliographic file. Unfortunately, it is in the nature of the beast that each problem can be met by a number of solutions and that system analysts have yet to agree on a best solution for any given problem.

PROCESSING

Three areas of processing must be provided by the CIS for bibliographic data bases. These are:

1) File maintenance,

2) Index production, and

3) Reference retrieval.

The first two are primarily associated with the internal operation of CIS, while the third is the major reason the library will acquire and process bibliographic data bases.

File Maintenance

This involves merging updated entries into a main file, deleting out-of-date entries, making corrections by replacing entries or items, etc. Also included in this area is the maintenance of any supporting files such as thesauri, or inverted files to be used for on-line processing.

Index Production

This involves production of printed indexes for the use of both the CIS analyst and the patron. Thesaurus listings with frequency of term use would be helpful in formulating search strategies. Some requests could be satisfactorily answered by referring to printed author, KWIC title or other indexes.

If the researcher is making a comprehensive literature search, the bibliography resulting from a machine search may number hundreds of entries. In this case, printed indexes by author, originating institution, index term, etc. or KWIC or KWOC title indexes would make the result more useful. (14)

REFERENCE RETRIEVAL

The subject of reference retrieval has been of central interest to the study of information science and documentation since its inception. The relevant chapters of the Annual Review of Information Science and Technology(1) provide excellent reviews of the topic with selected bibliographies.

In its simplest form, reference retrieval may be defined as the selection of a set of references from a larger collection according to some known criterion. Although this definition does not exclude manual systems, the following discussions deal primarily with computer oriented reference retrieval systems.

A common, though rather artificial, distinction is made between a retrieval system and a dissemination system. In a retrieval system, a bibliographic file is searched against a "query" to print a "bibliography", while, in a dissemination system, it is searched against a "user profile" to print a "selective dissemination of information (SDI) notice".(4) There is no logical difference between a "query" and a "user profile" or a "bibliography" and an "SDI notice". The former both represent a request for information, while the latter both represent the end product satisfying that request. The real difference is in the nature of the file searched. For retrieval, the file usually represents a relatively large historical collection (say, 2-3 years of Chemical Abstracts) used for a retrospective search, while for dissemination, the file usually is a current addition (say, the last issue of Chemical Abstracts) used for current awareness. Obviously, the computer makes no distinction between files of different sizes or between possible uses for its output. Thus a single computer program will perform retrieval or dissemination with equal facility.

Recognizing, then, the essential similarity between the two types of systems, they are referred to in this report by the common term reference retrieval. Also, the more meaningful terms query and bibliography will be preferred.

It is worth-while, at this time, to consider whether there are any advantages to the computerized searching of a bibliographic data base as opposed to the manual searching of the original printed source.

First, in regard to user behavior, Berul(4) has reviewed an evaluation of an on-line document retrieval system which ranks the importance of access points to a collection as: subject, author, report number, corporate source and contract number.

A printed indexing or abstracting journal will normally supply subject and author indexes and may or may not supply report number, corporate source, contract number or other indexes. With indepth

subject indexing, it may not be feasible or economical to print a complete subject index, consequently a citation may only be entered in the printed index under the two or three most important subjects of the nine or ten assigned to a document (see, for example, Index Medicus or Research in Education).

A subject search rarely involves a single index term but usually requires a combination of index terms, either to narrow the subject area or to include closely related index terms in the search. Such a search can be performed using a printed index but may be difficult and time consuming. Also, with only a partial printed index some relevant citations are almost certain to be missed.

The principle advantages of searching a bibliographic file by computer, then, occur in just those areas where a printed source is weak. To the computer, the entire citation is accessible. In principle, it is no more difficult for a computer to examine all index terms than just the major ones, or to examine the author(s), corporate source, publication date or any other access point contained in the citation. The computer will also search tirelessly for complex coordinations of index terms or other access points, missing nothing it has been asked for.

The above relatively abstract justification of computer searching of bibliographic files is amply supported by both the increased availability of bibliographic files in machine readable form and the number of reference retrieval systems that are being used or designed across the nation. (12)

From the computer's viewpoint, there are at least five elements to a query:

1) Identification,

2) File specification,

3) Selection specification,

4) Sequence specification, and

5) Output specification.

The query is normally presented to the computer in a precise form specified by a "query language" (in a programming sense) for matching against a file.

The identification information is used merely to enable people to correlate the output with a query and to simplify further processing. It may be as simple as a query number, or as complex as a sentence summarizing the request plus the requester's name and address.

The file specification states the file to be searched. In a system designed to search only one file, this is normally implied. In some cases, a portion of a file may be specified as, say, certain years or issues or as a range of accession numbers. Such a requirement may be handled by the program or implied by the file given to the program to be searched in a particular run.

The selection specification is the heart of the matter. Essentially, it is a statement of the conditions an entry in the file must satisfy in order to be selected and output by this query. Commonly used conditions include the presence of a given author, the presence of given index terms or keywords in the title or abstract, the presence of a citation to a given article, etc.

While the query languages for retrieval systems are no more standardized than file formats, most have provisions for specifying conditions of the form:

<Item> <Relation> <Value>

where Item is a field in an entry (e.g. author), Value is a potential value that Item can have (e.g. Smith, John), and Relation specifies the particular condition desired between the Item and Value (e.g. =, ¬=, >, etc.). In addition, the Boolean operators (And, Or and Not) are normally available for specifying logical combinations of condition e.g.:

Condition_1 OR Condition_2 AND NOT Condition_3.

The particular form, punctuation, spelling, etc. required by the query language varies from system to system, depending on the capabilities of the system and the ingenuity of the designer. An extremely simple query language is given in Program Input Data (Section III).

The sequence specification states the order desired for the entrie of the bibliography. Some possibilities are: alphabetic order by the first author, descending order by the publication date or descending order by the degree of match (when selection conditions are weighted). Often, a given retrieval system will allow only an implied sequence, i.e. entries are printed in the sequence given in the file.

The output specification normally states the items that are to be printed from each entry and how to print them. In many cases, a given retrieval system will have a standard output format which is used for all queries, or will allow a selection from a few standard formats (e.g. with abstract or without abstract).

The complete query with all the above elements, either explicit or implied, is then one set of input to a computer program for reference retrieval, the other set being the bibliographic file.

Computer oriented reference retrieval systems come in two basic types: batch-processing systems and interactive systems.

In a batch-processing system, a set of queries is collected into a "batch" and processed as a group against the file(s) desired. The file is normally stored as a sequential file on magnetic tape. As each entry in the file is read, it is compared against the selection specification for each query in the batch. If the entry satisfies the conditions for a given query, it is written into a temporary file together with an identification for the query. When the complete file has been processed, the work file is sorted to bring the "hit" references for a given query together. This sorted file is printed as a bibliography for each query.

In an interactive system, there is a dialog between the patron or analyst and the computer through a typewriter-like terminal or a television-like (CRT) terminal. The computer can then provide assistar in formulating a query, estimate the magnitude of the results, perform the search, allow a review of retrieved references for relevance, automatically modify the query on the basis of user relevance judgemen and print the resultant bibliography, all in a few minutes. Each query is usually processed as a separate entity, even though several queries may be processed simultaneously through different terminals. The file is normally stored on magnetic disk or some other direct access storage device in a file organization designed to provide rapid response to queries. (15)

These descriptions of batch-processing and interactive systems are general. As is so often the case, there are many variations on a basic theme. One interesting hybrid results when interaction is used to formulate a query which is then saved for a scheduled batch mode search. There are, of course, advantages and disadvantages to each type of system, with probably the most important considerations being cost, response time and feasibility within a particular environment.

II. APPROACHES TO REFERENCE RETRIEVAL SOFTWARE

There are a number of alternatives available for the design of a reference retrieval system for the Center for Information Services. In examining some of these, the reader should keep in mind that any system must be capable of processing a number of data bases in a wide variety of formats, not necessarily at the same time or with the same program.

CUSTOM PROGRAMMING

This approach involves writing or obtaining an individual program for retrieval from each data base. Programs of this type are available from the originating agencies, e.g. Chemical Abstracts Service or the Library of Congress, or from other investigators, e.g. Syracuse Universi School of Library Science.(3) Any programs obtained from other organizations for a computer similar to the CIS computer would probably have to be modified to some extent to resolve any differences in the configurations, operating systems or programming language translators. Many programs would have to be written from scratch, either due to insurmountable differences between existing programs and the CIS computer or to a lack of any program suitable for the needs of the CIS.

A solution based only on these techniques would lead to a large number of programs to be used and maintained. Worse yet, there would be a number of different query languages that the CIS analyst would have to be familiar with in order to properly formulate queries. In general, this approach is almost certain to be too lengthy, too costly and unresponsive to the needs of the CIS to be considered for anything other than experimentation or very short term use.

DATA BASE CONVERSION

This approach involves defining a canonical format for bibliograph files and converting each data base acquired into this format. Individual programs would then have to be written, of course, to convert each data base. However, a single reference retrieval program could be used to search the canonical form of the data bases.

While the conversion programs would be relatively simple and the CIS analyst would have only one query language to deal with,

the computer costs incurred and probable loss of information during the conversion process argue strongly against this approach.

MODULAR PROGRAMMING

This approach depends upon the nature of a reference retrieval program. Recall that there are two basic types of input: a set of queries and a bibliographic file. Certain parts of such a program will have to "know" the structure of the query language either in the external or in the internal form (e.g. a query translation routine), other parts will have to "know" the format of the bibliograph. file (e.g. a file read routine), some parts must deal with both (e.g. a matching routine), and some are concerned with neither (e.g. a monitor routine). The designer of a modular program would recognize these essential differences and would isolate into separate procedures those portions of the program which are concerned with a specific type of information.

One possible segmentation of a reference retrieval program for searching one file is as follows:

1) Main procedure - knows other principal procedures and calls them in the proper order.

2) Query Input procedure - knows the gross external and internal forms of the query, but does not know the detailed structures of the selection, sequence, and output specifications. Calls the query translation procedure.

3) Query Translation procedure - knows the detailed external and internal forms of the selection, sequence and output specifications. Performs translation from external to internal form.

4) File Read procedure - knows the external format of the bibliographic file. Reads the file entries and extracts from them the information required for the matching process. Is called by the matching procedure.

5) Matching procedure - knows the internal form of the selection specification and the information extracted from the file entries. Performs the matching of the two and writes matched entries into a work file.

6) Sort procedure - knows the internal form of the sequence specification. Sorts matched entries into the proper sequence.

7) Print procedure - knows the external file format and interr output specification. Reads the sorted f'le of matched entries and prints the bibliographies.

Some of the implications of this organization are interesting. First, let us assume that the output is to be in a standard format for all queries. This means that there is no need for any form of output specification. Now, notice that only the File Read and Print procedures know the external format of the file. Under the above assumption, both of these procedures are relatively straightforward. It is an easy task, then, to write a File Read and a Print procedure for a different file. Substituting these new procedures for the corresponding old ones, would allow essentially the same program to search the second file.

The modular program presented in Section III is an elaboration on the above outline. While it was designed with several purposes in mind, it is an experimental program to evaluate the overall desig approach. Those familiar with programming will recognize that it is a giant step from a reference retrieval system discussed in general terms, as in this paper, to an actual working program. Taking that giant step involves endless obstacles, evaluations of alternatives, decisions, dead ends, reevaluations of objectives, etc. Much of the value of this process is the knowledge gained in the actual doing.

Some decisions, however, must be backed up by experimentation a measurement. To that purpose this program has been designed to enab experimentation with different file organizations. In particular, i can search a sequential or an inverted file, and a sequential file can be searched for either a Boolean combination of "subject access" terms or a list of "accession" numbers (where the definition of what constitutes a subject access term or an accession number is implied in the File Read procedure). This allows the testing of several alternative modes of operation.

In one mode, the inverted file would be searched for a subject to produce a list of potentially relevant accession numbers. This list could then be used to refer to a printed form of the file. Alternatively, this list could be used to search the sequential file to produce a bibliography. Yet a third alternative would be to sear for the subject against the sequential file to begin with. Very rea questions need to be answered as to which mode or modes are cost/ effective and/or acceptable from the viewpoint of the users of the C system. In the author's opinion, all or most possibilities should b optional. A part of the process of formulating a query would involv deciding which search mode to use. The program is designed to enabl other file organizations to be implemented in as painless a manner a possible.

Since the Cis environment postulates the existence of many bibliographic files, this program is designed to search any number of these files, either inverted or sequential, in a single computer run. The program automatically determines which files and organizations are requested by the queries and searches only those requested. Although the primary purpose of the program has been as an operational system on which to base the development of the various procedures (acquisitions, cataloging, retrieval, etc.) attendant to the operation of a CIS, it has proven useful as a classroom and demonstration search tool.

GENERALIZED DATA BASE MANAGEMENT SYSTEMS

The computer industry, particularly those elements concerned with business data processing, has realized for many years that much of the work associated with creating and updating files, retrieving information from files, sorting the retrieved information (as needed), and printing reports from it, is highly repetitious.

Modular programs on a par with the outline described above can do much to help the situation but tend to be rather specific to the problem at hand. With the goal of solving a class of problems rather than just one member of the class, a large number of generalized data base (or file) management systems have been developed.

The key concept of such a system is the separation of the external format of the file from the program. To illustrate, for the modular program outline, if the external format of the file can be presented to the File Read and Print procedures as a third type of input, the program becomes "generalized" in the sense that it can process any file that can be described to it. The description of a file for the program only needs to be done once (unless the file format changes) and can then be saved and used as needed. The set of file descriptions at an installation may be maintained in a central directory or dictionary, providing the additional benefit of improved management control, or, alternatively, they may be associated directly with the file itself making the file, in effect, self-describing.

A complete generalized data management system will also include provisions for creating and maintaining the files and file descriptions either through additional procedures or separate programs. The information contained in a query, perhaps now better termed a processing request, would of course be expanded to include provisions for controlling these functions.

One indication of the success of this approach is the large number of these systems that have been developed by individual computer installations, the independent software firms and the computer manufacturer

In 1959, McGee(17) described a set of generalized file handling routines developed at General Electric. Today, there are probably more than one hundred of these systems. Many of these have been designed for a specific environment or with limited goals, e.g. QWICK QWERY(7) which is only a retrieval system, but the number of true generalized systems is still large.

The literature of generalized data base management systems falls into several categories. Those items concerned with a specific system, a technique or an application will be largely ignored here The interested reader is referred to the reviews and bibliographies in the Annual Review. (1) However, a number of surveys do deserve mention here.

The recent CODASYL report(6) presenting features of the ADAM, GIS, IDS, ISL-1, MARK IV, NIPS/FFS, SC-1, TDMS and UL/1 systems is one of the most important in the field, not only for its breadth and depth of coverage but also for its largely successful attempt to translate the terminology used by the various systems into a standard terminology (for instance, the terms Item, Property, Field and Element are all used in at least one of the above systems for what is called an Item by this paper). This effort should be expanded to other systems.

Sundeen(18) describes the evolution of general purpose software and gives short summaries of the GIS, MANAGE, BEST, IMRADS, INFOL, TDMS and ASI-ST systems. Byrnes and Steig(5) present a list of features useful for discriminating between systems and summarize the AEGIS, ASI-ST, CDMS, GIM, INFORMS, GIS-BASIC, and MARK IV systems. One feature, price, ranges from less than $15,000 for AEGIS to about $100,000 for CDMS. Ziehe(21) describes the external and internal data structuring and organization for the TDMS, RFMS, DM-1, GIS and CATALOGS systems and compares these features for the various systems.

Rather than explicitly comparing features between systems, the System Development Corporation(10) has defined a benchmark data base management problem patterned after a business organization chart and personnel file. Possible solutions to this problem are presented for the COLINGO D, COLINGO C-10, ADAM, MARK III, and the Massachusetts General Hospital systems, and approaches to the problem are given for the BEST and IDS systems.

Reilly(19) has examined the CFSS, INFOL, MARK IV and GIS management systems and compared them specifically from the viewpoint of processing bibliographic files.

As part of a session on file management systems at the ASIS (American Society for Information Science) annual meeting in October 1969, an exercise in computer file management was defined and participants were invited to send solutions. The exercise was

patterened after a document retrieval system and was designed to examine the functional capabilities of various systems. The exercise and solutions using the TIP, UL/1, CAPRI and TEXT-PAC systems and one using PL/1 together with three general purpose systems have been published. (2)

In the face of this alphabet soup of generalized systems, what can be said about their suitability for the CIS? As a class, these systems are in the unfortunate position of being both too general and not general enough for efficient searching of bibliographic files.

Recall that these systems are mainly designed for use in a business environment. A business has control over the format of its own files and can manipulate them to conform to the capabilities of a given system; moreover, business files are composed largely of fixed formats of fixed length items although variable length items and repeating items are used to some extent.

The CIS has no control over the formats of the files it acquires and, while most systems are capable of handling any file format that can be described to them, none of them are capable of describing the range of bibliographic files discussed earlier.

A normal part of these systems is the preparation of business reports, particularly in responding to an ad hoc request. For this purpose, they usually include extensive capabilities in the sequence and output specification areas. These capabilities are wasted in reference retrieval where a standard sequence and a standard output format or a limited number of standard formats is perfectly adequate. Moreover, a business report is normally a tabular arrangement of short data items. A bibliographic citation with a thousand character abstract hardly fits into this arrangement. Thus the generality of the sequence and output specifications represents, on the one hand, unnecessary inefficiency and, on the other, inadequate capability.

Also, while file maintenance, retrieval and reporting are normal operations for a generalized data base management system, the remaining CIS requirement, index production, is not a normal business operation and is not within the capabilities of most, if not all, such systems.

Finally, many of these systems tend to be relatively inefficient in terms of the amount of computer time used. This is due to the fact that they operate using a processing request and a file description as data. This data must be interpreted each time an entry is examined or compared with a selection specification or printed. Interpreting the data each time it is needed is a duplication of effort in that the same information is derived each time it is done. More advanced systems such as ASI-ST (by Applications Software, Inc.) improve efficiency by translating the processing requests and/or file

descriptions into machine instructions or subroutine linkages or by generating source language code for subsequent translation into machine instructions. In any case, executing the compiled instructions is more efficient than interpretive execution.

The author is certainly not the possessor of any unique insights into these problems with generalized data base management systems, nor is the computer industry standing on its past performance. Primarily in response to the more complex data base description requirements posed by the burgeoning technology in business management information systems, a Special Interest Committee on File Description and Translation (SICFIDET) has been formed in the Association for Computing Machinery and is actively investigating the problems and some of their possible solutions. This group overlaps in membership with the Ad Hoc Committee on Data Definition Languages (DDL's) which recently reported(11) to the X3 Committee on Computers and Information Processing of the American National Standards Institute (ANSI, formerly U.S.A. Standards Institute). This report briefly summarized the needs, current state and future problems of Data Definition Languages and made recommendations of what needs to be standardized and what actions X3 should take to insure orderly progress in the future.

A report(8) by the Data Base Task Group to the CODASYL Programming Language Committee is probably the most complete and unified effort in this area to date. This report defines a Data Description Language (DDL) to be used for describing data bases and a Data Management Language (DML) to be used for processing data bases. The DML is not a complete programming language, but is proposed as an extension of COBOL.

In addition to these formal efforts in data definition, the originators of generalized data base management systems are constantly at work improving their products and pushing back the boundaries of the state-of-the-art. On balance, the future developments of generalized data base management systems deserve to be watched with the idea of applying these systems to the processing requirements of the Center for Information Services. Indeed, the needs of the CIS can and should influence the future capabilities of these systems.

CIS FILE MANAGEMENT SYSTEM(19)

This approach involves combining the best features of file management systems with parameterized special purpose modules to create a generalized system to meet the special requirements of the Center for Information Services.

The heart of this system would be a fairly conventional generalized file management system (dubbed the CISFMS for short). The major

features that distinguish the CISFMS from most of the other generalized systems that have been discussed include:

1) A more general capability in the file description. All of the file formatting techniques discussed earlier should be accommodated plus any others found by examining additional currently available files. The systems should be able to accommodate new formatting techniques with a minimum of difficulty as file designers come up with new solutions to problems.

2) Execution from compiled machine instructions. This requirement is necessary for efficient processing. In addition, the system must be able to define, save and utilize standard modules such as file read routines or print routines.

3) The ability to interface with special purpose subsystems for more complex processing of files. There would be one subsystem for processing bibliographic files, one for statistical files and one for natural language files.

The usual file management operations including file maintenance, simple entry selection, sorting and report generation would be performed by the CISFMS. Operations like index production, analysis of variance or concordance generation would be passed on to the appropriate subsystem.

The overall design of the CISFMS would be modular to permit interchangeable components such as file read procedures and to allow the system to be implemented and expanded in rational stages.

III. A MODULAR PROGRAM FOR REFERENCE RETRIEVAL

GENERAL

This computer program is designed to provide subject access and/or accession number access to a wide variety of the magnetic tape files of indexing, abstracting and cataloging information created and distributed by various agencies.

The files that can be searched are essentially limited to those of the bibliographic type. Currently, the only file this program is capable of using is the ERIC file produced as a by-product of the Research in Education abstracting journal published by the Educational Resources Information Center. This was chosen as the initial test file because it was readily available to the author.

In its present form the program is suitable for retrospective and selective dissemination searches of the ERIC file.

In broad terms the program can be divided into three phases:

1) Query processing,
2) File searching, and
3) Result printing.

A query contains an identification, the patron's name, and the name and type of the file to be searched. The query statement can be either a Boolean expression of the subject access terms representing an area of interest or a list of the accession numbers of the specific entries desired.

The file to be searched can be of either a sequential or an inverted type. For the sequential type, each entry represents one document (article, report, monograph, etc.). The accession number and subject access terms are imbedded in the entry and are extracted by a read routine for that file for matching against queries. For the inverted type, each record represents one subject access term and consists of a list of the accession numbers of all entries containing that term. Selected records are processed to produce a list of accession numbers satisfying a given query.

A file search is performed for each set of queries requesting a given file name and type, thus more than one file and/or file type can be searched in a single computer run.

The result of a search is either a set of hit entries, for a search against a sequential file, or a list of accession numbers, for a search against an inverted file. These are printed as a bibliography or list of accession numbers for each query by a print routine written for the particular file searched.

In addition, task CPU time and elapsed time are generated and printed for the following operations:

1) Reading and translating each query,
2) Searching an inverted file for each query,
3) Searching a sequential file for all queries requesting that file, and
4) Printing the results of each query.

The program is written entirely in PL/I with the exception of a short Assembler language procedure used to measure task CPU time and elapsed time.

PROGRAM NARRATIVE

The program consists of eleven external procedures logically organized as shown in the Program Procedure Chart (See Figure 1). They are:

FILESCH	MATCHS
READQ	ERICRS
TRANSQE	PRINTH
ERICTQL	ERICPH
SEARCHF	STIMER
MATCHI	

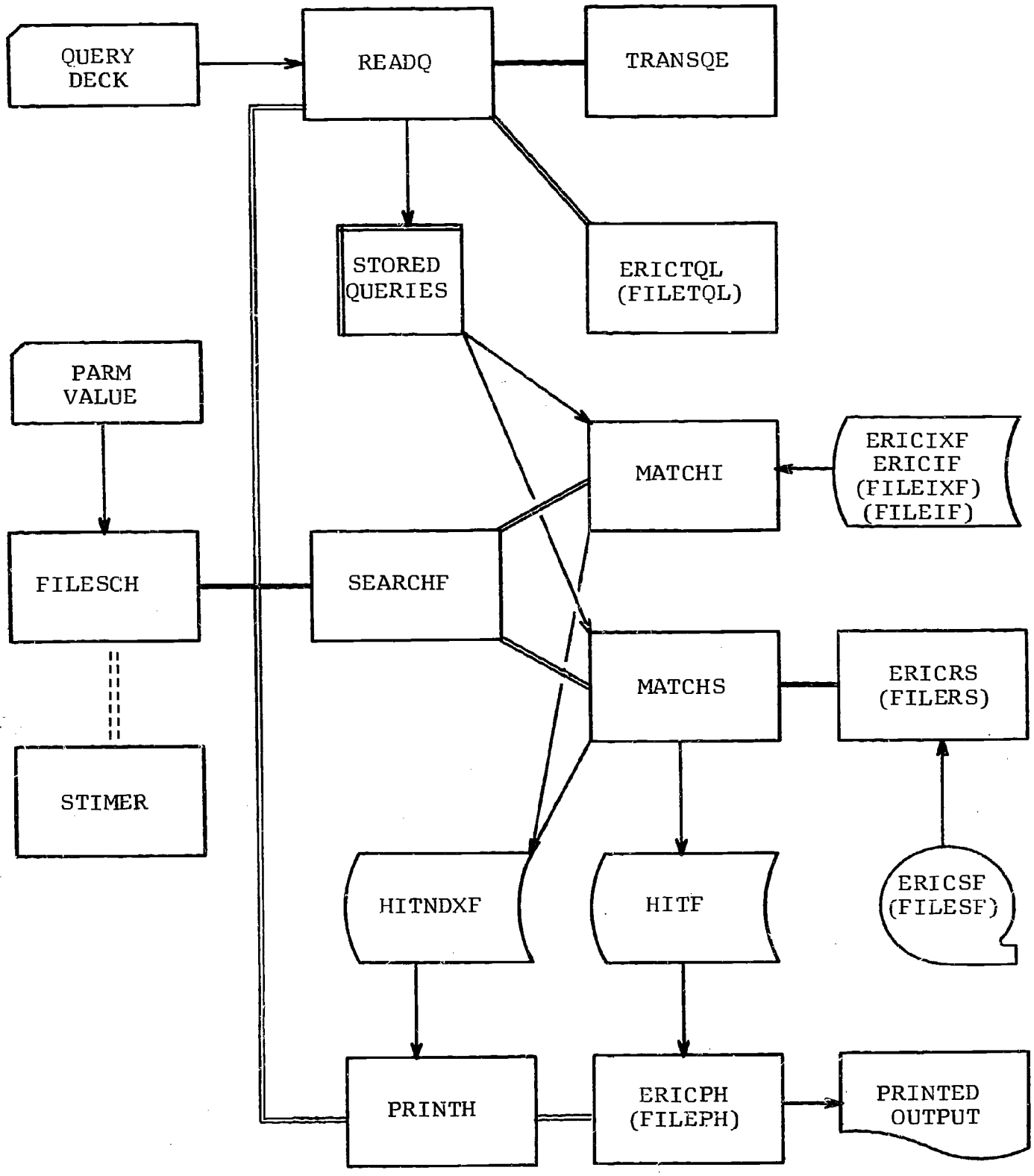
FILESCH is the main procedure. It prints an opening message, determines the maximum number of queries that can be processed for this run and allocates storage for control information on each query. It then calls in turn the READQ, SEARCHF and PRINTH procedures to do the real work.

The READQ procedure reads the query deck, one query at a time, determines the type of query, and calls either the TRANSQE procedure to translate an expression form query or the ERICTQL procedure to translate a list form query. If the translation was successful, READQ will allocate storage for the query, store it and fill in the control information for the query.

The TRANSQE procedure translates an expression form query from infix notation to Polish postfix notation for efficient processing by the match procedures.

FIGURE 1

PROGRAM PROCEDURE CHART



The ERICTQL procedure translates a list form query into an array of binary accession numbers for use by the MATCHS procedure.

The SEARCHF procedure partitions the total set of queries into subsets whose members all require the same file and file type. For each subset, it will call either the MATCHI procedure to search an inverted file, or the MATCHS procedure to search a sequential file. After all files to be searched have been searched, the storage occupied by the query expressions and lists is freed for reuse.

The MATCHI procedure processes queries from the current subset, one by one, by reading accession number lists from an inverted file (ERICIF and ERICIXF files) and matching or merging them as specified by the query to produce the final list of hit accession numbers. This list is then written into the HITNDXF file.

The MATCHS procedure obtains one entry at a time from the sequential file by calling the ERICRS procedure. Each entry is matched against all queries from the current subset. If a match occurs with any query, the entry is copied into the HITF file. In addition, a short entry is written into the HITNDXF file for each query that entry matched.

The ERICRS procedure reads one entry from the sequential file (ERICSF file) each time it is called, and extracts from the entry a binary accession number (ED number) and an array of subject access terms (Descriptors). These are then returned to the MATCHS procedure for comparison against the queries.

The PRINTH procedure reads and stores the entries from the HITNDXF file for use by the ERICPH procedure. It then calls the ERICPH procedure once for each query to print the results for that query.

The ERICPH procedure scans the stored hit index for the entries belonging to the current query. It will then print either a bibliography from the entries located via the index and read from the HITF file, or a list of ED numbers directly from the index.

The STIMER procedure measures elapsed CPU time and real time. It has been placed as shown in the chart to avoid unnecessary complication of the chart. It is actually called by the READQ, MATCHI, MATCHS and PRINTH procedures which use it to obtain the times printed for the operations mentioned above.

Notice that ERICTQL, ERICRS and ERICPH are the only procedures dependent upon the format of the ERIC file. A different file could be searched by adding analogous procedures (FILETQL, FILERS and FILEPH) as shown on the chart. A FILETQL procedure would translate

list form queries for the file. An inverted file (FILEIF and FILEIXF) can be searched by the MATCHI procedure once it has been created in the proper format. A FILERS procedure would read a sequential file (FILESF) and supply the proper information to the MATCHS procedure. A FILEPH procedure would print either a bibliography, or an accession number list as the results of a query. The FILETQL and FILERS procedures are required only if their functions are desired. In addition, the READQ, MATCHS and PRINTH procedures would have to undergo minor changes to recognize the existence of the new file and procedures.

PROGRAM INPUT DATA

The program input is a card deck (hereafter termed query deck) which is read from the standard PL/I SYSIN file.

The query deck is composed of a number of individual queries. Each query consists of:

1) A query control card, and
2) A query statement.

The program will normally allow up to 25 queries to be processed in a single run. This number can be modified, to a maximum of 150, by supplying the appropriate PARM value to the program.

Query Control Card

The query control card provides information identifying the query, and the name and type of the file to be searched. The format is as follows:

```
$$id        QUERY name, file name, file type
```

1) $$ — This must appear in columns 1 and 2 to identify the card as a control card.

2) id — This is an arbitrary identification for the query. From 0 to 8 characters may be punched in columns 3 to 10.

3) QUERY — This must be punched in columns 12 to 16 to identify the card as a query control card.

4) name — This is the name of the person for whom the query is being prepared. Only the first 24 characters will be used. A comma separates this field from the file name.

5) file name This is the 4 character name of the file to be searched. Currently, the only valid file name is "ERIC". If this field is in error or omitted, the default file name "ERIC" will be supplied.

6) file type This is the 4 character type of the file to be searched. Allowable values are:

a) INVT - Inverted file, and
b) SEQN - Sequential file.

If this field is in error or omitted, "INVT" will be supplied as a default for an expression form query statement, or "SEQN" for a list form query statement.

The query control card may not be continued to a second card and may not contain significant information past column 72. Columns 73 to 80 may be used for sequence numbering.

Query Statement

The query statement can span as many as 50 cards. The statement may be punched anywhere in columns 1 to 72. The first column of the second or subsequent cards is considered to follow column 72 of the previous card. Columns 73 to 80 may be used for sequence numbering.

The query statement itself may be in one of two forms:

1) Expression form, or
2) List form.

Expression Form Query Statement. The expression form of the query statement is written as a Boolean combination of "subject access" terms (subject headings, index terms, descriptors, etc.).

Subject access terms appear in the expression enclosed in single quotation marks, e.g. 'DATA PROCESSING', and must be punched exactly as they would appear in the file, including blanks, punctuation, etc.

The allowable Boolean operators are:

1) & (And),
2) | (Or),
3) ¬ (Not).

Parentheses may be used freely to group terms and the relationships between them. Blanks may be used outside of terms to improve readability.

If parentheses are not present to explicitly indicate grouping, the priority of operators is identical to that of PL/I, i.e. ¬ (highest), &, | (lowest). This would cause the following expression:

'TERM A'|¬'TERM B'&'TERM C'

to be interpreted by the program as meaning:

'TERM A' |((¬'TERM B')&'TERM C')

All entries of the file whose subject access terms satisfy the conditions of the query expression will be selected as hits.

This form of the query statement may be used with both file types. For the INVT type, the output will be a list of matching accession numbers. For the SEQN type, the output will be a bibliography of matching entries.

Sample expression form queries:

```
$$QUERY1   QUERY DR. SMITH, ERIC, INVT

('INTELLIGENCE TESTS')&('TEST VALIDITY' | 'TEST RESULTS')

$$ENGINSTR QUERY MARK CASEY, ERIC, SEQN

'ENGLISH INSTRUCTION'&¬'REMEDIAL PROGRAMS'
```

List Form Query Statement. The list form of the query statement is written as a list of accession numbers (LC catalog card numbers, ED report numbers, etc.) of the entries desired. Since the form of an accession number depends on the file, the following discussion applies to the ERIC file only.

For the ERIC file, the accession number is termed an ED number. Each ED number is written as a six to eight character field, e.g. ED014397. The first two characters must be "ED", the remaining characters must be numeric. ED numbers must be separated by commas. Blanks may be used to improve readability. ED numbers need not be in ascending order.

This form of the query statement may only be used with the SEQN file type. If the INVT type is specified, it will be changed to SEQN. The output will be a bibliography of the specified entries.

Sample list form query:

$$QUERY1 QUERY DR. SMITH, ERIC, SEQN

ED010372, ED010373, ED012292, ED013655, ED015273, ED010374

PROGRAM FILES

The major files of interest are identified as the Inverted ERIC File, the Sequential ERIC File, and the Hit Files.

Inverted ERIC File

This file is in reality a pair of files. The first, the ERICIF file, is the actual inverted file. The second, the ERICIXF file, is an index to the ERICIF file to provide rapid direct access to its records.

The ERICIF file is a PL/I REGIONAL(3) keyed direct access file of unblocked variable length records. The record key is the descriptor term (subject access term). The record itself consists of a count field followed by a list of all the ED numbers (accession numbers) of the resumes containing that descriptor term. The count field contains the number of entries in the list. Both the count field and list entries are four byte binary integers. The first record of this file has an all blank key and contains the ED numbers of all resumes in the file. This record is used in processing Boolean $\neg$ (Not) operators. The remaining records are in ascending alphabetic order by descriptor term. Within each record, the ED numbers are also in ascending order.

A record is read from this file by specifying the key (descriptor term) of the record desired and a relative track number of the file to begin looking for the record. The specified track and successive tracks will be searched until the record is found or until a specified (LIMCT) number of tracks has been searched without finding the record. Obviously, records are read most efficiently if the track specified contains the record desired.

The ERICIXF file is designed to make this possible. This file is a sequential file of blocked fixed length entries which operates as a track index to the ERICIF file. Each entry is fifty bytes long. The first ten bytes of the first entry contain, in character form, the number of the last track in the ERICIF file. The remaining entries of the file contain the last descriptor term written on each track of the ERICIF file.

Thus, when a given term record is to be read from the ERICIF file, the ERICIXF file (stored in core) is first examined to determine the particular track of the ERICIF file to scan for the record.

The pair of files described above is directly analogous to an IBM Indexed Sequential file organization with an in-core master index. That organization, however, is designed to operate with fixed length records only.

While this inverted file design is efficient to read, maintenance is quite a different matter (and one not discussed in this report). The interested reader is referred to Reference 15.

Sequential ERIC File

This file is a sequential file of blocked variable length entries. Each entry represents one report resume.

Each entry is composed of a number of variable length groups and each group has the following structure:

| LENGTH | TAG | DATA FIELD... |

The LENGTH is two bytes long and contains the binary character count for the entire group (including the LENGTH and TAG).

The TAG is two bytes long and contains a binary number which specifies the type of information in the Data Field.

The Data Field is LENGTH-4 bytes long and contains the significant information of the group, usually in character form.

Figure 2 contains the TAGs in use and the corresponding Data Fields. The TAG values are given in decimal.

Not all fields are contained in each entry, although TAGs 0, 1, and 2 appear to be mandatory. Many fields (e.g. DESC) have subfields which are delimited by a semicolon (;). See reference 20 for additional information.

Hit Files

Two hit files are used to retain the results of the file search(es) before they are printed. These files are very similar in logical organization to the inverted file.

The HITF file is a PL/I REGIONAL(3) keyed direct access file of unblocked variable length records. The record key is the accession

FIGURE 2

ERIC FILE DATA FIELDS

TAG	Keyword	Data Field Name
0	none	Sequence
1	none	Add Date
2	none	Change Date
16	ACC	Accession Number
17	CH	Clearinghouse Accession Number
18	OCH	Other Clearinghouse Accession Number
20	PA	Program Area
23	PDAT	Publication Date
26	TITL	Title
27	AUTH	Personal Author
28	INST	Institution Code
32	SPON	Sponsoring Agency Code
35	DESC	Descriptors
36	IDEN	Identification
37	PRICE	EDRS Price
38	NOTE	Descriptive Note
43	ISS	Issue
44	ABST	Abstract
45	REPNO	Report Number
46	CONT	Contract Number
47	GR	Grant Number
48	BN	Bureau Number
49	AVAIL	Availability
50	JNL	Journal Citation
128	INSTNM	Institution Name
132	SPONNM	Sponsoring Agency Name

number (in character form) of the following hit entry, which is a direct copy from the sequential file. Only one copy of the hit entry is written in this file regardless of the number of queries the entry matched. This file is written by the MATCHS procedure.

The HITNDXF file is a sequential file of blocked fixed length entries. Each entry has the following form:

| REC# | TRACK# | QUERY# |

where REC# is an accession number, TRACK# is a relative track number in the HITF file, and QUERY# is a query number. REC# is a four byte binary integer, while TRACK# and QUERY# are two byte binary integers. There is one entry in this file for each query-entry match.

Entries are written by both the MATCHS and the MATCHI procedures. The entries written by the MATCHS procedure contain the track number of the complete hit record in the HITF file, which is used with the accession number (converted to character form) to read the hit record itself. For those entries written by the MATCHI procedure, the track number is not used (set to -1) and the accession number is printed directly from the entry. The query number is used to select those entries used to print the results for a given query. These files are deleted after the program is completed.

PROGRAM OUTPUT

The program output is written on the standard PL/I SYSPRINT file. Figure 3 illustrates all normal output. The following comments are keyed to the numbers circled on this figure.

1) The program name, date run and maximum number of queries are printed by FILESCH.

2) The query and related messages are printed by READQ.

3) These messages are printed by SEARCHF.

4) The trace for the inverted file search is printed by MATCHI.

5) The trace for the sequential file search is printed by MATCHS.

6) The search results are printed by ERICPH.

7) The print times are printed by PRINTH.

FIGURE 3

SAMPLE OUTPUT

(Part 1 of 4)

```
MULTIPLE FILE SEARCH PROGRAM                          DATE:  29 JUN 1970   (1)
MAXIMUM NUMBER OF QUERIES:  25

QUERY ID:  EXPRINVT  NAME:  AEINT DE BOER
$$EXPRINVT QUERY AEINT DE BOER,ERIC,INVT                          ADB00100   (2)
'LANGUAGE ARTS'&'SCHOOL IMPROVEMENT'&                             ADB00120
(¬'ENVIRONMENT')                                                  ADB00140
QUERY TRANSLATION SUCCESSFUL.
CPU TIME:        .009 SECS.  ELAPSED TIME:        .011 SECS.

QUERY ID:  EXPRSEQN  NAME:  AEINT DE BOER
$$EXPRSEQN QUERY AEINT DE BOER,ERIC,SEQN                          ADB00160   (2)
'CURRICULUM PLANNING'&'ELEMENTARY EDUCATION'                      ADB00180
QUERY TRANSLATION SUCCESSFUL.
CPU TIME:        .006 SECS.  ELAPSED TIME:        .008 SECS.

QUERY ID:  LISTSEQN  NAME:  AEINT DE BOER
$$LISTSEQN QUERY AEINT DE BOER,ERIC,SEQN                          ADB00200   (2)
ED001003, ED001017, ED001023                                      ADB00220
QUERY TRANSLATION SUCCESSFUL.
CPU TIME:        .004 SECS.  ELAPSED TIME:        .006 SECS.
```

FIGURE 3 (Continued)

SAMPLE OUTPUT

(Part 2 of 4)

```
FILE SEARCH TRACE.                                                    ] (3)


FILE NAME:  ERIC  FILE TYPE:  INVT                                    ] (4)
QUERY ID:  EXPRINVT  NAME:  AEINT DE BOER
HITS:      3
CPU TIME:        .008 SECS.  ELAPSED TIME:        .287 SECS.


FILE NAME:  ERIC  FILE TYPE:  SEQN                                    ] (5)
      24 RECORDS READ.
QUERY ID  NAME                          HITS
EXPRSEQN  AEINT DE BOER                    2
LISTSEQN  AEINT DE BOER                    3
CPU TIME:        .071 SECS.  ELAPSED TIME:        .508 SECS.


END OF FILE SEARCH.                                                   ] (3)
```

FIGURE 3 (Continued)

SAMPLE OUTPUT

(Part 3 of 4)

```
QUERY ID:  EXPRINVT  NAME:  AEINT DE BOER   HITS:  3   PAGE 1

  ED001006  ED001018   ED001022


CPU TIME:        .003 SECS.  ELAPSED TIME:          .010 SECS.


QUERY ID:  EXPRSEQN  NAME:  AEINT DE BOER   HITS:  2  PAGE 1


ED001007
FASAN, WALTER R. ; AND OTHERS
A UNIT FOR THE CLASSROOM TEACHER--THE NEWSPAPER, A LIVING
AND DYNAMIC TEXTBOOK.  (TITLE SUPPLIED)
CHICAGO PUBLIC SCHOOLS, ILL.
PUB DATE:  60.
EDRS PRICE MF-$0.09 HC-$0.88, 21P.
   DESCRIPTORS:  CURRICULUM PLANNING: *ELEMENTARY EDUCATION;
*NEWS MEDIA; *SLOW LEARNERS; SUPPLEMENTARY EDUCATION;
*TEACHING GUIDES.



ED001013
BEACON PROGRAM.
FORD FOUNDATION, NEW YORK, N.Y.; PHILADELPHIA PUBLIC
SCHOOLS, PA.
PUB DATE:  60.
EDRS PRICE MF-$0.09 HC-$1.92, 45p.
  DESCRIPTORS:  *COMMUNITY INVOLVEMENT; *CULTURAL ENRICHMENT;
*CULTURALLY DISADVANTAGED; *CURRICULUM PLANNING; EDUCATIONAL
OBJECTIVES; ELEMENTARY EDUCATION; FAMILY SCHOOL
RELATIONSHIP; SCHOOL IMPROVEMENT.


CPU TIME:        .020 SECS.  ELAPSED TIME:          .130 SECS.
```

⑥ ⑦ ⑥ ⑦

FIGURE 3 (Continued)

SAMPLE OUTPUT

(Part 4 of 4)

QUERY ID: LISTSEQN NAME: AEINT DE BOER HITS: 3 PAGE 1 ⑥

ED001003
KORNHAUSER, LOUIS H.
THE LANGUAGE ARTS PROJECT PROGRAM.
PUB DATE: SEP 63.
EDRS PRICE MF-$0.09 HC-$0.88, 22P.
DESCRIPTORS: *CULTURALLY DISADVANTAGED; ELEMENTARY SCHOOLS; *KINDERGARTEN; *LANGUAGE ENRICHMENT; PARENT ATTITUDES; *SPEECH EDUCATION; TEACHER WORKSHOPS.

ED001017
SOME UNIQUE FEATURES.
PUB DATE: 01 NOV 62.
EDRS PRICE MF-$0.09 HC-$0.12, 2P.
DESCRIPTORS: BILINGUALISM; *COMMUNITY PROGRAMS; CULTURAL ENRICHMENT; CULTURALLY DISADVANTAGED ; *INSERVICE TEACHER EDUCATION; *LANGUAGE ARTS; LITERATURE PROGRAMS; *REMEDIAL PROGRAMS; TEACHER EDUCATION; *TUTORING.

ED001023
HOLMES, ELIZABETH ; AND OTHERS
POLICIES AND PROCEDURES FOR THE SELECTION OF TEXTBOOKS IN GREAT CITIES.
PUB DATE: 15 NOV 63.
EDRS PRICE MF-$0.09 HC-$0.56, 12P.
DESCRIPTORS: *EDUCATIONAL NEEDS; *TEXTBOOK CONTENT; *TEXTBOOK EVALUATION; *TEXTBOOKS; *TEXTBOOK SELECTION; TEXTBOOK STANDARDS.

CPU TIME: .028 SECS. ELAPSED TIME: .090 SECS. ⑦

END OF PROCESSING. ①

The sequential file used for this illustration consists of 24 entries with the abstracts deleted. The inverted file used was created from this sequential file.

PROGRAM EXECUTION COSTS

The following figures were derived from various tests run on the IBM 360 Model 91 at the Campus Computing Network at UCLA. The cost figures reflect the charging algorithm at CCN, which is:

$$\$ \text{ COST} = .15(T + .02I)\ (1 + .004R)$$

where T is the CPU time used in seconds,

I is the number of I/O operations, and

R is the region requested in 1K units.

(The above formula is accurate only for programs requesting 250K bytes or less.)

The test runs were all made against a sample file of 1202 ERIC entries or the corresponding inverted file.

Test Parameters:

Test	Queries	Query Type	File Type	Terms/ Acc#s	Hits
1	11	Expression	INVT	33	173
2	11	Expression	SEQN	33	173
3	11	List	SEQN	174	173

Test Cost Statistics:

Test	CPU Time	I/O Count	Region Requested	Region Used	$ Cost
1	.99s	284	130K	118K	$ 1.52
2	11.45s	1015	150K	138K	$ 7.62
3	8.61s	1021	150K	138K	$ 6.97

CONTINUED PROGRAM DEVELOPMENT

While this documentation represents a complete, operational, and useful program, it must be recognized that a program of this nature is a dynamic entity. Continued use and review will suggest desirable additions, new techniques to improve efficiency or

generality, or unused features. It is useful to consider some of the changes and/or extensions the program may undergo in the future.

Improving Efficiency

Let us adopt cost as a measure of efficiency. Reducing the cost then becomes synonymous with improving efficiency. Now, in view of the charging formula at CCN, there are three factors determining the cost:

1) Region size requested,
2) Total I/O count, and
3) Total CPU seconds.

Reducing any one of these three, without an offsetting increase in the others will result in a reduction of cost.

One simple method of reducing the region size is to make more efficient use of core storage by creating an overlay program structure. In this manner, procedures which do not need to be present in storage at the same time can reuse the same areas of storage.

Examination of the program structure and the procedure lengths indicates that a 40 to 50% reduction in region size can be accomplished by this means, with a resulting program size of, say, 70K to 80K. Offsetting inefficiency can result if the same procedure has to be loaded into storage for execution more than once. This situation can be avoided by suitable arrangement of the query deck and/or automatic optimization of the order of procedure use. It is interesting to note that an overlay structure would allow the program to be used on a computer as small as a 128K IBM 360 Model 40.

A point where region size and I/O count come into conflict is the area of file blocking. Increasing the blocking factor of a file will generally reduce the I/O count and simultaneously increase the region used by the file. The number of I/O buffers assigned to a file will also affect the amount of storage used. To minimize cost, the proper balance must be maintained between these factors.

In order to reduce the CPU time it is instructive to examine the times measured by the STIMER procedure for the test runs presented under Program Execution Costs.

Procedure	Test 1		Test 2		Test 3	
	CPU	Real	CPU	Real	CPU	Real
READQ	.071	.129	.077	.318	.157	.737
MATCHI	.192	3.297				
MATCHS			7.004	30.347	4.238	27.358
PRINTH	.095	.242	3.774	33.649	3.557	26.664
Overhead	.63s		.59s		.66s	
Totals	.99s		11.45s		8.61s	

It is obvious that the greatest benefits are to be realized in reducing the time spent searching the sequential file with expression form queries (this used about 23% of the Model 91 CPU in Test 2). Relatively large amounts of CPU time are also spent searching the sequential file with list form queries and in printing the bibliographies. These activities involve the MATCHS, ERICRS and ERICPH procedures.

It is a well known fact that higher level programming languages, such as PL/I, rarely produce machine code comparable in efficiency to that which can be written in Assembler language. To illustrate this point, the innermost loop of the MATCHS procedure was written first in "good" PL/I, with an eye on minimizing the number of PL/I statements executed. With this code, Test 2 used over 61% of the Model 91 CPU in the matching phase (24.840 seconds of CPU time and 40.354 seconds of real time). Projecting these timing values to a smaller computer produced enormous search times. A look at the machine code produced by PL/I revealed glaring inefficiencies due both to inadequate optimization on the part of the compiler (Version 5) and the use of language features requiring extensive code and/or numerous PL/I library procedure calls.

Experimentation with different means of restating the same computations produced dramatic reductions in the amount of processing. As a benchmark a query of the form:

('A'|'B'|'C'|'D')&('E'|'F'|'G'|'H')&¬('I'|'J')

and an entry containing ten descriptors (none of which match any term in the query) was used. To match one query against one entry, a reduction in processing from 4,757 machine instructions including 193 subroutine calls to 1,415 machine instructions including 10 subroutine calls was noted. This is in the neighborhood of a 70 to 90% reduction. The interested reader is invited to make assumptions for a number of queries, number of entries, and average subroutine length and compute a figure for the number of instructions avoided.

Then consider the time it would require a given computer to execute them. Then consider that the same match could be performed in about 550 machine instructions in Assembler language.

Thus, it is easy to see that further tuning of the program can be expected to decrease both CPU time and region size, a double benefit. Rewriting any given procedure in Assembler language can be expected to save 50% or more in both CPU time and region size. Rewriting the entire program in Assembler language should make it possible to run on a 65K or perhaps even a 32K IBM 360 Model 30.

Even in PL/I a significant reduction in cost might be achieved by modifying the matching algorithm. Three possibilities come to mind.

The first depends on the nature of the Boolean "And" and "Or" operators. Take for example, an expression like 'A'&'B'. If 'A' is known to be false (i.e. 'A' does not appear in the current entry), there is no need to check for 'B', for the entire expression is false. Similarly, in the expression 'A'|'B', if 'A' is true, then the entire expression is true. Eliminating redundant comparisons of subject access terms against the entry will reduce expense to the extent that a comparison costs more than recognizing that it is redundant.

The second possibility could reduce the cost by rapidly eliminating a large percentage of the entries as not matching any query. The approach would be to build a master list, in alphabetical order, of all the terms in the queries. The index terms for a given entry, also in alphabetical order, would be matched against this list. If there is no match, the entry could be rejected for all queries. If there is a match, each query would be matched individually against the entry. If the result of the "prematch" were saved in the master term list, this approach would have the added advantage of checking the entry for a given term only once even if it appears in more than one query.

The third possibility lies in superimposed coding.(13) In this approach, all the terms of a query would be combined into a single coded term with some loss of information. The same would be done to the index terms of an entry. If the single comparison of these coded terms for a query and an entry indicated a possible match, a more detailed comparison would be performed as before. Extending this idea further, a single master coded term could be created by combining the coded terms for the individual queries.

It is entirely possible to visualize the matching process as a succession of screens, of finer and finer grade. The first screen would compare the master coded term for all queries against the coded term for the entry. The second would compare the master term

list against the index terms for the entry. The third would compare the coded terms for each query against the coded term for the entry. Finally, the fourth would evaluate the expression for a query.

Of course, each screen will have an associated cost in terms of CPU time and storage. But each will also reject from further consideration an entry or query which does not pass its test, and eliminate the cost of more detailed processing. Just which screens are economically justifiable is not intuitively obvious.

If sufficient disk space were available, a sequential file search could be avoided altogether by using the inverted file as an entry to a direct access copy of the complete file.

Another alternative would be to automatically use the output from an inverted file search, an accession number list, as the input for a search of the sequential file.

Both of these last two approaches run into the offsetting costs of creating, storing and maintaining the various disk files. The inverted file deserves particular mention here. Currently, this file is stored at about 35% efficiency in terms of the actual amount of information stored compared to the disk space required to store it. In addition, this file is limited to about 1800 entries by the length of the "Not"-list and the length of a 2316 disk track. Both of these conditions are undesirable, and should be eliminated by redesign. Any redesign of this file should also take into consideration the need to maintain the file.

Extensions

There are potential capabilities for improvement other than efficiency that might prove desirable.

One possibility for extension is the matching process. Currently, an exact match is required between a query term and any entry subject access term. The exact match condition could be augmented by various types of term truncation, term weights and threshold logic, or additional relational operators. The matching process could also be extended to other fields of the entry, e.g. author, publisher, publication date, etc.

Another suitable area for extension is the output. At this time, each file has its own print procedure with a fixed print format. This might be extended to allow selection from a number of print formats or to allow a new file of hit entries to be written. Further extension might enable the requester to specify his own print formats, possibly

by the use of a generalized print procedure for all files. In a similar vein, the same concept could be carried over to a generalized read routine for all files.

It goes without saying that the form and processing of the queries would have to undergo modification to provide for additional information specified by the requester; and it must also be recognized that each additional measure of generality will add its own penalty in terms of efficiency and cost. If the program as a whole is rewritten into Assembler language, features that improve operating system limitations should almost certainly be incorporated at the same time.

One problem that must be considered in an operational context is reliability. No matter how well written a program is, there is always a chance that a system disturbance, malfunction, or deficiency can force a run to be terminated prematurely. A run with several hundred queries and several thousand entries to be searched may represent an expense of hundreds of dollars and hours of computer time. Thus it is highly desirable to be able to recover from an abnormal termination and continue from a point at or near where the program left off. To do this, the operating system can provide assistance in obtaining program checkpoints (periodically saving program status) and restarting a program from a checkpoint. Alternatively, special procedures may be built into the program to restart execution in the middle of a file search or before or during the printing of the bibliographies (the most costly points for a failure). However, it may be more practical to create special versions of the program for restart purposes.

In summary, there are many possible directions in which program development can proceed. Some are complementary, some are contradictory, but all require careful consideration and possibly experimentation before they become a reality.

IV. SUMMARY

Bibliographic, statistical and natural language machine readable data bases are becoming a recognized source of information.

Just as the library has adapted its operations in the past from clay tablets, to scrolls, to books and to films, records, maps, microfiche and other non-book materials, it must adapt to this new type of material.

The Center for Information Services is proposed as a unit within a university research library to provide the necessary administrative, technical and public services to handle these data bases. The degree of success of the CIS will depend to a large extent on the computerized services that it can provide. These services depend, in turn, upon the capabilities of the computer systems, both hardware and software.

The final software system for the CIS can only be defined in rather gross functional terms currently. As the CIS passes through the successive stages of planning, design, programming, implementation and operation, its software will most likely evolve from experimental custom programming products to modular programs for specific functions to a special purpose file management system.

The modular program for reference retrieval as developed for CIS experimentation and as described in this report is one more step in what could be a long, hard journey, but a journey that must be completed if the library is to adequately meet users' needs.

BIBLIOGRAPHY

1. American Documentation Institute, Annual Review of Information Science and Technology. Carlos A. Cuadra, ed., New York, Interscience, 1966-.

   Individual chapters provide state-of-the-art reviews in specific areas of information science. Includes extensive bibliographies. In 1968, the ADI changed its name to the American Society for Information Science.

   Beginning with Volume 3 (1968) publication was under the new name by Encyclopaedia Britannica. Ann W. Luke became the assistant editor for Volume 4 (1969).

2. ASIS: Journal of the American Society for Information Science, Vol. 21, No. 3, May-June 1970.

   The six articles of interest in this issue are:

   1) Climenson, W. Douglass, "An Exercise in Computer File Management", p. 201-203.
   2) Mathews, William D., "Using the TIP System in the ASIS File Management Exercise", p. 204-208.
   3) Higgins, Timothy and Mays, Robert, "A Solution to the ASIS File Management System Exercise Using PL/I and Three General Purpose Systems", p. 204-213.
   4) Olle, T. William and Gagnoud, Andre M., "A Solution to the ASIS File Management Exercise Using RCA's UL/1", p. 214-218.
   5) Bloom, P.W., "Application of CAPRI to the ASIS File Management Exercise", p. 219-223.
   6) Lindsley, Thomas F., "Application of IBM TEXT-PAC to the ASIS File Management Exercise", p. 224-227.

3. Atherton, Pauline and Miller, Karen B. "LC/MARC on MOLDS: An Experiment in Computer-Based, Interactive Bibliographic Storage, Search, Retrieval and Processing", Journal of Library Automation, Vol. 3, No. 2, June 1970, p. 142-165.

Describes the Management On-Line Data Systems (MOLDS) as used to process MARC data. The system is written in FORTRAN for the IBM 360 in both batch and interactive versions. MARC file is reformatted into fixed format entries. The file is apparently searched sequentially.

4. Berul, Lawrence H. "Document Retrieval", Annual Review of Information Science and Technology, Vol. 4, Chicago, Encyclopaedia Britannica, 1969, p. 203-227.

Reviews recent developments in document retrieval, in particular computerized retrieval systems. Notes the shift in the literature from batch processing to interactive systems. 80 references.

5. Byrnes, Carolyn J. and Steig, Donald B. "File Management Systems: A Current Summary", Datamation, Vol. 15, No. 11, November 1969, p. 138-142.

6. CODASYL Systems Committee, A Survey of Generalized Data Base Management Systems, New York, ACM, May 1969, 398 p.

7. Consolidated Analysis Centers, Inc. QWICK QWERY User's Manual, Santa Monica, Calif., 1968, paged by part.

Covers processing requests for the QWICK QWERY system from the user's point of view.

8. Data Base Task Group Report to the CODASYL Programming Language Committee, New York, ACM, October 1969, 191 p.

9. de Boer, Aeint H. A Modular Program for Reference Retrieval from Bibliographic Data Bases in a University Research Library, University of California, Los Angeles, 1970, 174 p. (MSIS Thesis)

10. "Five Approaches to the Same Data Base Problem", Proceedings of the Second Symposium on Computer-Centered Data Base Systems, Santa Monica, Calif., System Development Corp., 1 December 1965, p. 3-3 to 3-275. (TM-2624/100/00).

11. Gosden, John A. "Report to X3 on Data Definition Languages", FDT: Journal of the ACM SICFIDET, Vol. 1, No. 2, December 1969, p. 14-25.

12. Housman, Edward M. "Survey of Current Systems for Selective Dissemination of Information (SDI)", Proceedings of the American Society for Information Science, Vol. 6, Westport, Conn., Greenwood, 1969, p. 57-61.

Briefly tabulates the results of a survey of 100 SDI operations. The "typical" operation serves 50-500 users from externally supplied bibliographic files. Profiles are "typically" prepared by a professional analyst using a thesaurus and Boolean logic.

13. Hutton, Fred C. "PEEKABIT, Computer Offspring of Punched Card PEEKABOO, for Natural Language Searching", Communications of the ACM. Vol. 11, No. 9, September 1968, p. 595-598.

Describes the use of superimposed codes in searching natural language text obtained from the subject indexes of Nuclear Science Abstracts. The technique is also applicable to fixed vocabularies.

14. Jordan, John R. and Watkins, W. J. "KWOC Index as an Automatic By-Product of SDI", Proceedings of the American Society for Information Science, Vol. 5, New York, Greenwood, 1968, p. 211-215.

SDI notices judged as "of interest" by the user are automatically accumulated to produce a bibliography, author index and KWOC title index every thirteen weeks.

15. Lefkovitz, David. File Structures for On-Line Systems, New York Spartan, 1969, 215 p.

Discusses the role of file structure in on-line interactive systems with particular attention to search and maintenance. Covers multilist, inverted, controlled length multilist and cellular file organizations.

16. Library of Congress, Information Systems Office, MARC Manuals Used by the Library of Congress, Chicago, American Library Association, 1969, paged by part.

This publication combines four MARC system manuals. The first, "Subscriber's Guide to the MARC Distribution Services", is the manual of specifications for magnetic tapes in the MARC II format.

17. McGee, William C. "Generalization: Key to Successful Data Processing", Journal of the ACM, Vol. 6, No. 1, January 1959. p. 1-23.

Describes an early system for the IBM 702 consisting of generalized sort, file maintenance and report generator routines.

18. Sundeen, Donald H. "General Purpose Software", Datamation, Vol. 14, No. 1, January 1968, p. 22-27.

19. University of California, Institute of Library Research, Mechanized Information Services in the University Library. Phase I: Planning, Los Angeles, 15 December 1967, paged by part. (PB-178 441, PB-178 442).

This thirteen part study is the final report to the National Science Foundation on Phase I of the CIS project. The parts referenced in the body of this paper are:

Part 3. Troutman, Joan C. "Inventory of Available Data Bases", 57p.
Part 5. Reilly, Kevin D. "Nature of Typical Data Bases", 51p.
Part 6. Reilly, Kevin D. "Evaluation of Generalized File Management Systems", 45p.
Part 10. "Preliminary Specifications (Hardware and Software) for a Center for Information Services", 43p.

20. Worth, J. M. et.al. ERIC Information Processing, Storage and Retrieval: System and Program Documentation, Anaheim, California, North American Rockwell Corp., August 1968, paged by part.

This is a report to the U.S. Office of Education with the documentation of the computer system designed for the Educational Resources Information Center.

21. Ziehe, Theodore W. Data Management: A Comparison of System Features, Austin, Texas, TRACOR, October 1967, 43p. (AD-661 861).