MARC PROGRAM RESEARCH AND DEVELOPMENT: A PROGRESS REPORT

Henriette D. AVRAM, Alan S. CROSBY, Jerry G. PENNINGTON, John C. RATHER, Lucia J. RATHER, and Arlene WHITMER: Library of Congress, Washington, D. C.

A description of some of the research and development activities at the Library of Congress to expand the capabilities of the MARC System. Gives details of the MARC processing format used by the Library and then describes programming work in three areas: 1) automatic tagging of data elements by format recognition programs; 2) file analysis by a statistical program called GENESIS; and 3) information retrieval using the MARC Retriever.

The MARC System was designed as a generalized data management system that provides flexibility in converting bibliographic descriptions of all forms of material to machine readable form and ease in processing them. The foundation of the system is the MARC II format (hereinafter simply called MARC), which reached its present form after many months of planning, consultation, and testing. Implementation of the system itself has required development of a battery of programs to perform the input, storage, retrieval, and output functions necessary to create the data base for the MARC Distribution Service.

These programs are essentially like those of the MARC interim system described in the report of the MARC pilot project (1). Briefly, they perform the following tasks:

1) A pre-edit program converts records prepared on an MT/ST to a magnetic tape file of EBCDIC encoded record segments.
2) A format edit program converts the pre-edited tape file to a modified form of the MARC processing format.
3) A content edit program generates records in the final processing format. At this stage, mnemonic tags are converted to numeric form, subfield codes may be supplied, implicit fixed fields are set, etc.
4) IBM SORT program arranges validated content-edit output records by LC card number. This program is also used later in the processing cycle.
5) A generalized file maintenance program (Update 1) allows addition, deletion, replacement, or modification of data at the record, field, or subfield levels before the record is posted to the master file. A slightly different version (Update 2) is used to update the master file.
6) A print index program generates a list of control numbers for a given file. The list may also include status, date of entry, or date of last transaction for each record.
7) A general purpose print program produces a hardcopy to be used to proofread the machine data against the original input worksheet. Since the program is table controlled, it can be modified easily to yield a great variety of other formats and it can be extended routinely to handle other data bases in the MARC processing format.
8) Two additional programs select new records from the MARC master file and convert them from the processing format to the communications format on both seven- and nine-track tapes for general distribution.

As the basic programs became operational, it was possible to investigate other aspects of the MARC System that would benefit from elaboration and refinement. Reports of some of this activity have found their way into print, notably a description of the MARC Sort Program and preliminary findings on format recognition (2, 3), but much of the Library's research and development effort in programming is not well known. The purpose of this article is to give a progress report on work in three significant areas: 1) automatic tagging of data elements by format recognition programs; 2) file analysis by a statistical program called GENESIS; and 3) information retrieval using the MARC Retriever.

In the following descriptions, the reader should bear in mind that all of the programs are written to accommodate records in the MARC processing format. A full description of the format is given to point up differences between it and the communications format. All of the programs are written in assembly language for the IBM S360/40 functioning under the disk operating system (DOS). The machine file is stored on magnetic tape and the system is operated in the batch mode.

At present, the programs described here are not available for general distribution, but it is expected that documentation for some of them may be filed with the IBM Program Information Department in the near future. Meanwhile, the Library of Congress regrets that it will be unable to supply more detailed information. It is hoped that the information in this article will answer most of the questions that might be asked.

MARC PROCESSING FORMAT

The MARC data base at the Library of Congress is stored on a nine-channel magnetic tape at a density of 800 bpi. The file contains records in the undefined format; each record is recorded in the MARC processing format (sometimes called the internal format). Data in the processing format are recorded in binary, packed decimal, or EBCDIC notation depending on the characteristics of the data and the processing required. The maximum length of a MARC processing record is 2,048 bytes. The magnetic tape labels follow the proposed standard developed by Subcommittee X3.2 of the United States of America Standards Institute.

A MARC record in the processing format is composed of six parts: record leader (12 bytes), communications field (12 bytes), record control field (14 bytes), fixed fields (54 bytes), record directory (variable in length, with each directory entry containing 12 bytes) and variable data fields (variable length). All records are terminated by an end-of-record (EOR) character.

Record Leader

0 1	2 4			5	6	7	8	9 11
Record length	Date			Status	Not used	Record type	Bibliographic level	Not used
	YY	MM	DD					

Element number	Name	Number of characters	Character position in record	Definition
1	Record length	2	0-1	Total number of bytes in the logical record including the number of bytes in the record length itself. It is given in binary notation.
2	Date	3	2-4	Date of last transaction (i.e., the date the last action was taken upon the whole record or some part of the record). The date is recorded in the form of

				YYMMDD, with each digit being represented by a four-bit binary-coded decimal digit packed two to a byte.
3	Status	1	5	A code in binary notation to indicate a new, deleted, changed, or replaced record.
4	Not used	1	6	Contains binary zeros.
5	Record type	1	7	An EBCDIC character to identify the type of record that follows (e.g., printed language material).
6	Bibliographic levels	1	8	An EBCDIC character used in conjunction with the record type character to describe the components of the bibliographic record (e.g., monograph).
7	Not used	3	9-11	Contains binary zeros.

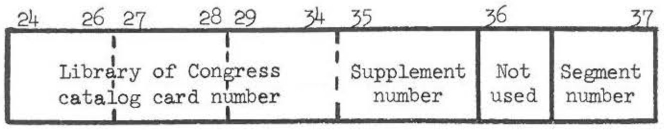
Communications Field

12 13	14 15	16	17	18	19	20 23
Record directory location	Directory entry count	Record source	Record destination	In-process type	In-process status	Not used

Element number	Name	Number of characters	Character position in record	Definition
1	Record directory location	2	12-13	The binary address of the record directory relative to the first byte in the record (address zero).
2	Directory entry count	2	14-15	The number of directory entries in the record, in binary notation. There is one directory entry for every variable field in the record.
3	Record source	1	16	An EBCDIC character to show the cataloging source of the record.

4	Record destination	1	17	An EBCDIC character to show the data bank to which the record is to be routed.
5	In-process type	1	18	A binary code to indicate the action to be performed on the data base. The in-process type may signify that a new record is to be merged into the existing file; a record currently in the file is to be replaced, deleted, modified in some form; or that it is verified as being free of all error.
6	In-process status	1	19	A binary code to show whether the data content of the record has been verified.
7	Not used	4	20-23	Contains binary zeros.

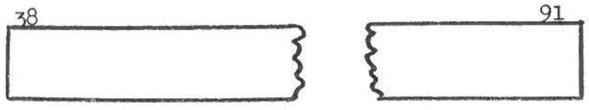
Record Control Field



Element number	Name	Number of characters	Character position in record	Definition
1	Library of Congress catalog card number	12	24-35	On December 1, 1968, the Library of Congress initiated a new card numbering system. Numbers assigned prior to this date are in the "old" system; those assigned after that date are in the "new" system(4). The Library of Congress catalog card number is always represented by 12 bytes in EBCDIC notation but the data elements depend upon the system.

	Old numbering system			
	Prefix	3	24-26	An alphabetic prefix is left justified with blank fill; if no prefix is present, the three bytes are blanks.
	Year	2	27-28	
	Number	6	29-34	
	Supplement number	1	35	A single byte in binary notation to identify supplements with the same LC card number as the original work.
	New numbering system			
	Not used	3	24-26	Contains three blanks.
	Initial digit	1	27	Initial digit of the number.
	Check digit	1	28	"Modulus 11" check digit.
	Number	6	29-34	
	Supplement number	1	35	See above.
2	Not used	1	36	Contains binary zeros.
3	Segment number	1	37	Used to sequentially number the physical records contained in one logical record. The number is in binary notation.

Fixed Fields



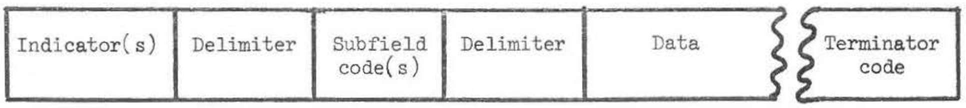
The fixed field area is always 54 bytes in length. Fixed fields that do not contain data are set to binary zeros. Data in the fixed fields may be recorded in binary or EBCDIC notation, but the notation remains constant for any given field.

Record Directory

92 94	95	96 98	99	100 101	102 103
Tag	Site number	Not used	Action code	Data length	Relative address

Element number	Name	Number of characters	Character position in record	Definition
1	Tag	3	92-94	An EBCDIC number that identifies a variable field. The tags in the directory are in ascending order.
2	Site number	1	95	A binary number used to distinguish variable fields that have identical tags.
3	Not used	3	96-98	Contains binary zeros.
4	Action code	1	99	A binary code used in file maintenance to specify the field level action to be performed on a record (i.e., added, deleted, corrected, or modified).
5	Data length	2	100-101	Length (in binary notation) of the variable data field indicated by a given entry.
6	Relative address	2	102-103	The binary address of the first byte of the variable data field relative to the first byte of the record (address zero).
7	Directory end of field sentinel	1	n	Since the number of entries in the directory varies, the character position of the end-of-field terminator (EOF) also varies.

Variable Data Fields



Element number	Name	Number of characters	Character position in record	Definition
1	Indicator	Variable	n	A variable data field may be preceded by a variable number of EBCDIC characters which provide descriptive information about the associated field.
2	Delimiter	1	n	A one-byte binary code used to separate the indicator(s) from the subfield code(s). When there are no indicators for a variable field, the first character will be a delimiter.
3	Subfield code	Variable	n	Variable fields are made up of one or more data elements (5). Each data element is preceded by a delimiter; a lower-case alphabetic character is associated with each delimiter to identify the data element. These alpha characters are grouped. All variable fields will have at least one subfield code.
4	Delimiter	1	n	Each data element in a variable field is preceded by a delimiter.
5	Data	Variable	n	
6	Terminator code	1	n	All variable fields except the last in the record end with an end-of-field terminator (EOF); the last variable field ends with an end-of-record terminator (EOR).

FORMAT RECOGNITION

The preparation of bibliographic data in machine readable form involves the labeling of each data element so that it can be identified by the machine. The labels (called content designators) used in the MARC format are tags, indicators, and subfield codes; they are supplied by the MARC editors before the data are inscribed on a magnetic tape typewriter. In the current MARC System, this tape is then run through a computer program and a proofsheet is printed. In a proofing process, the editor compares the original edited data against the proofsheet, checking for errors in editing and keyboarding. Errors are marked and corrections are reinscribed. A new proofsheet is produced by the computer and again checked for errors. When a record has been declared error-free by an editor, it receives a final check by a high-level editor called a verifier. Verified records are then removed from the work tape and stored on the master tape.

The editing process in which the tags, indicators, subfield codes, and fixed field information are assigned is a detailed and somewhat tedious process. It seems obvious that a method that would shift some of this editing to the machine would in the long run be of great advantage. This is especially true in any consideration of retrospective conversion of the 4.1 million Library of Congress catalog records. For this reason, the Library is now developing a technique called "format recognition." This technique will allow the computer to process unedited bibliographic data by examining the data string for certain keywords, significant punctuation, and other clues to determine the proper tags and other machine labels. It should be noted that this concept is not unique to the Library of Congress. Somewhat similar techniques are being developed at the University of California Institute of Library Research (6) and by the Bodleian Library at Oxford. A technique using typographic cues has been described by Jolliffe (7).

The format recognition technique is not entirely new at the Library of Congress. The need was recognized during the development of the MARC II format, but pressure to implement the MARC Distribution Service prevented more than minimal development of format recognition procedures. In the current MARC System a few of the fields are identified by machine. For example, the machine scans the collation statement for keywords and sets the appropriate codes in the illustration fixed field. In general, however, machine identification has been limited to those places where the algorithm produces a correct result 100 percent of the time.

The new format recognition concept assumes that, after the unedited record has been machine processed, a proofsheet will be examined by a MARC editor for errors in the same way as is done in the current MARC System. Since each machine processed record will be subject to human review, it will be possible to include algorithms in the format recognition program that do not produce correct tagging all of the time.

The format recognition algorithms are exceedingly complex, but a few examples will be given to indicate the nature of the logic. In all the examples, it is assumed that the record is typed from an untagged manuscript card (the work record used as a basis for the Library of Congress catalog card) on an input device such as a paper tape or a magnetic tape typewriter. The data will be typed from left to right on the card and from top to bottom. The data are input as fields, which are detectable by a program because each field ends with a double carriage return. Each field comprises a logical portion of a manuscript card; thus the call number would be input as a single field, as would the main entry, title paragraph, collation, each note, each added entry, etc. It is important to note that the title paragraph includes everything through the imprint.

Identification of Variable Fields

Call Number.

This field is present in almost every case and it is the first field input. The call number usually consists of 1-3 capital letters followed by 1-4 numbers, followed by a period, a capital letter, and more numbers. There are several easily identifiable variations such as a date before the period or a brief string of numbers without capital letters following the period.

The delimiter separating the class number from the book number is inserted according to the following five-step algorithm:

1) If the call number is LAW, do not delimit.
2) If the call number consists simply of letters followed by numbers (possibly including a period), do not delimit. Example: HF5415.13 If this type of number is followed by a date, it is delimited before the blank preceding the date. Example: HA12‡ 1967
3) If the call number begins with 'KF' followed by numbers, followed by a period, then:
    a) If there are one or two numbers before the period, do not delimit. Example: KF26.L354 1966a
    b) If there are three or more numbers before the period, delimit before the last period in the call number.
    Example: KFN5225‡.Z9F3
4) If the call number begins with 'CS71' do not delimit unless it contains a date. In this case, it is delimited before the blank preceding the date. Example: CS71.S889‡ 1968
5) In all other cases, delimit before the last capital letter except when the last capital letter is immediately preceded by a period. In this latter case, delimit before this preceding period.

Examples:	PS3553.E73‡W6	E595.F6‡K4 1968
	PZ10.3.U36‡Sp	TX652.5‡.G63 1968

Name Main Entry.

The collation statement is the first field after the call number that can be easily identified by analyzing its contents. The field immediately preceding the collation statement must be the title paragraph. If there is only one field between the call number and the collation, the work is entered under title (tagged as 245) and there is no name main entry. If there are two or three fields, the first field after the call number is a name main entry (tagged in the 100 block). When three fields occur between the call number and collation, the second field is a uniform title (tagged as 240).

Further analysis into the type of name main entry and the subfield code depends on such clues as location of open dates (1921-), date ranges covering 20 years or more (1921-1967), identification of phrases used only as personal name relators (ed., tr., comp.), etc. The above clues strongly indicate a personal name. Identification of an ordinal number preceded by punctuation and a blank followed by punctuation is strongly indicative of a conference heading.

In the course of processing, delimiters and the appropriate subfield codes are inserted. Subfield code "d" is used with dates in personal names; subfield code "e" with relators.

Example: MEPS‡de Smith, John,‡1902-1967,‡ed.

Analysis for Fixed Fields

Publisher is Main Entry Indicator.

This indicator is set when the publisher is omitted from the imprint because it appears as the main entry. The program will set this indicator whenever the main entry is a corporate or conference name and there is no publisher in the imprint statement. This test will fail in the case where there is more than one publisher, one of which is the main entry, but occurrences of this are fairly rare (less than 0.2 percent).

Biography Indicator.

Four different codes are used with this indicator as follows: A = individual autobiography; B = individual biography; C = collected biography or autobiography; and D = partial collected biography. The "A" code is set when 1) "autobiographical", "autobiography", "memoirs", or "diaries" occurs in the title statement or notes, or 2) the surname portion of a personal name main entry occurs in the short title or the remainder of the title subfields. The "B" code is set when 1) "biography" occurs in the title statement, 2) the surname portion of a personal name subject entry occurs in the short title or the remainder of the title subfields, or 3) the Dewey number contains a "B" or a 920. The "C" code is set when 1) "biographies" occurs in the title statement or 2) a subject entry contains the subdivision "biography." There appears to be no way to identify a "D" code situation. Despite this fact, the biography indicator can be set correctly about 83 percent of the time.

Implementation Schedule

Work on the format recognition project was begun early in 1969. The first two phases were feasibility studies based on English-language records with a certain amount of pretagging assumed. Since the results of these studies were quite encouraging, a full-scale project was begun in July 1969. This project is divided into five tasks. Task 1 consisted of a new examination of the data fields to see if the technique would work without any pretagging. New algorithms were designed and desk-checked against a sample of records. It now seems likely that format recognition programs might produce correctly tagged records 70 percent of the time under these conditions. It is possible that one or two fixed fields may have to be supplied in a pre-editing process.

Tasks 2 through 5 remain to be done. Task 2 will provide overall format recognition design including 1) development of definitive keyword lists, 2) typing specifications, 3) determination of the order of processing of fields within a record, and 4) description of the overall processing of a record. When the design is completed, a number of records will go through a manual simulation process to determine the general efficiency of the system design.

Task 3 will investigate the extension of format recognition design to foreign-language titles in roman alphabets. Task 4 will provide the design for a format recognition program based on the results of Tasks 2 and 3 with detailed flowcharts at the coding level. The actual coding, checkout, and documentation will be performed as Task 5. According to current plans, the first four tasks are scheduled for completion early in 1970 and the programming will be finished later in the year.

Outlook

It is apparent that a great deal of intellectual work must be done to develop format recognition algorithms even for English-language records and still greater ingenuity will be required to apply these techniques to foreign-language records. Nevertheless, on the basis of encouraging results of early studies, there is evidence that the human effort in converting bibliographic records to machine readable form can be materially reduced. Since reduction of human effort would in turn reduce costs, the success of these studies will have an important bearing on the rate at which current conversion activities can be expanded as well as on the economic feasibility of converting large files of retrospective cataloging data.

GENESIS

Early in the planning and implementation of automation at the Library of Congress it became apparent that many tasks require information about the frequency of data elements. For example, it was helpful to know about the frequency of individual data elements, their length in characters, and the occurrence of marks of punctuation, diacritics, and specified character strings in particular data elements. In the past, most of the counting has been done manually. Once a sizable amount of data was available in machine readable form, it was worthwhile to have much of this counting done by computer. Therefore, the Generalized Statistical Program (GENESIS) was done as a general purpose program to make such counts on all forms of material in the MARC Processing Format on magnetic tape files.

Any of a variety of counts can be chosen at the time of program execution. There are three types of specifications required for a particular run of the program: selection criteria; statistical function specifications; and output specifications.

Selection Criteria

Record selection criteria are specified by statements about the various data fields that must be present in the records to be processed. Field selection criteria specify the data elements that will actually be analyzed. Processing by these techniques operates logically in two distinct stages: 1) the record is selected from the input file; i.e., the program must determine if a particular record is to be included in the analysis; and 2) if the record is eligible, the specified function is performed on selected data fields. It should be noted that records may be selected for negative as well as positive reasons. The absence of a particular field may determine the eligibility of a record and statistical processing can be performed on other fields in the record. Record selection is optional; if no criteria are specified, all records on the input file will be considered for processing.

Since both record selection and field selection reference the same elements, specifications are input in the same way. Selection of populations can be designated by tagging structure (numeric tags, indicators, subfield codes or any combination of these three), specified character strings, and specified characters in the bibliographic data. The following queries are typical of those that can be processed by GENESIS. How many records with an indicator set to show that the volume contains biographic information also have an indicator set to show that the subject is the main entry? How many records with a field tagged to show that the main entry is the name of a meeting or conference actually have the words "meeting" or "conference" in the data itself? Table 1 shows the operators that can be used with record and field select statements.

Statistical Function Specification

The desired statistical function is specified via a function statement. Four functions have been implemented to date. They involve counts of occurrences of specified fields, unique data within specified fields given a range of data values, data within a specified range, and particular data characters. In addition to counting the frequency of the specified element, GENESIS calculates its percentage in the total population.

Table 1. Operators of GENESIS

Operator	*Example of usage*
EQUALS	Count all occurrences where data represented by tag 530 EQUALS "Bound with"
NOT EQUAL	Count all occurrences where the publication language code is NOT EQUAL to "eng"
GREATER THAN OR EQUAL TO	Count all occurrences and output records that are GREATER THAN OR EQUAL TO 1,000 characters
LESS THAN OR EQUAL TO	Count all occurrences of records entered on the MARC data base before June 1, 1968 (LESS THAN OR EQUAL TO 680601)
AND	Count all occurrences where the publication equals "s" AND the publication date is greater than or equal to 1960
OR	Count all occurrences of personal name main entry (tag 100) a relator (subfield code "e") that equals "ed." OR "comp."

The first function counts occurrences per record of specified field selection criteria. This answers queries concerning the presence of given conditions within the selected records; for example, a frequency distribution of personal name added entries (tag 700). This type of count results in a distribution table of the number of records with 0 occurrences, 1 occurrence, 2 occurrences, and so forth.

The second function, which counts occurrences of unique data values within a specified range, answers queries when the user does not know the unique values occurring in a given field, but can state an upper and lower value. For example, the specific occurrences of publishing dates between 1900 and 1960 might be requested. The output in response to this type of query consists of each unique value, lying within the range specified, with its frequency count. In addition, separate counts are given for values less than the lower bound and of values greater than the upper bound.

The function is performed by maintaining in computer memory an ordered list of unique values encountered, together with their respective counts. As selected fields are processed, each new value is compared against the entries in the list. If the new value already appears in the list, its corresponding count is incremented. Otherwise, the new value is inserted in the list in its proper place and the remainder of the list is pushed down by one entry. The amount of core storage used during a particular run is directly related to the number of unique occurrences appearing within the specified range. Since the length of each entry is determined by the length of the bounds specified, the number of entries which can be held in free storage can vary from run to run. Thus it is possible that the number of unique entries may fill memory before a run has been completed. When this happens, the value of the last entry in the list will be discarded and its count added to the "greater than upper bound" count. In this way, while the user may not obtain every unique value in the specified range, he will obtain all unique values from the lower bound which can be contained in memory. He is then in a position to make subsequent runs using, as a beginning lower bound value, the highest unique value obtained from the preceding run.

The third function processes queries concerning counts within specified ranges. When this function is used, unique values are not displayed. Instead, the occurrences are counted by specified ranges of values. More than one range can be processed during a single run. On output, the program provides a cumulative count of values encountered within each range as well as the counts of those less than and those greater than the ranges.

Function four counts occurrences of particular data characters. An individual character may be specified explicitly or implicitly as a member of a group of characters. This allows the counting of occurrences of various alphabetic characters within specified fields. The current list of character classes that can be counted are: alpha characters, upper-case letters, lower-case letters, numbers, punctuation, diacritics, blanks, full (all characters included in above classes), nonstandard special characters, and any particular character using hex notation. It should be noted that there are various ways of specifying particular characters. For example, an "A" might be designated causing totals to accumulate for all alphabetics; or, a "U" and an "L" might be specified causing separate totals to be accumulated for upper- and lower-case characters. In addition to the total counts for each class, individual counts of characters occurring within any class can be obtained for display along with the total count.

Output Specifications

Formatted statistical information is output to the line printer. Optionally, the selected records can be output on magnetic tape for later processing.

Limitations

For the purpose of defining a query, more than one field may be specified for record and field selection, using as many statements as necessary. At present, however, the statistical processing for a particular run is performed on all of the run-criteria collectively. For example, separate runs of the program are required to obtain each frequency distribution.

It is important to note that GENESIS is essentially a means of making counts. The statistical analysis of data is a complex task that requires sophisticated techniques. GENESIS does not have the capability to analyze data in terms of standard deviation, correlation, etc. but the output does constitute raw data for those kinds of analyses. Although the four functions of GENESIS implemented to date do not, in themselves, provide a complete statistical analysis, they greatly lessen the burden of counting; and techniques for designating data elements to be counted suffice to describe extremely complex patterns. Continued use of the program will no doubt provide guidelines for expansion of its functions.

Use of the Program

GENESIS has already provided analyses that are helpful in the design of automated procedures at the Library of Congress, as is indicated by the following instances. A frequency distribution of characters was made to aid in specifying a print train. An analysis of certain data characteristics has determined some of the specifications for the format recognition program described in an earlier section. GENESIS is providing many of the basic counts for a thorough analysis of the material currently being converted for the MARC Distribution Service to determine frequency patterns of data elements. The findings should be valuable for determining questions about storage capacity, file organization, and retrieval strategy. Although GENESIS is a new program in the MARC System, there is little doubt that it is a powerful tool that will have many uses.

MARC RETRIEVER

Since the MARC Distribution Service has been given the highest priority during the past two years, the emphasis in the implementation of the MARC System has been on input, file maintenance, and output with only minimum work performed in the retrieval area. It was recognized, moreover, that as long as MARC is tape oriented, any retrieval system put into effect at the Library of Congress would be essentially a research tool that should be implemented as inexpensively as possible. It did seem worthwhile, however, to build retrieval capability into the MARC System to enable the LC staff to query the growing MARC data base. Query capability would answer basic questions about the characteristics of the data that arise during the design phases of automation efforts. In addition, it seemed desirable to use the data base in an operational mode to provide some needed experience in file usage to assist in the file organization design of a large bibliographic data base.

The specifications of the system desired were: 1) the ability to process the MARC processing format without modification; 2) the ability to query every data element in the MARC record, alone or in combination (fixed fields, variable fields, the directory, subfield codes, indicators); 3) the ability to count the number of times a particular element was queried, to accumulate this count, print it or make it available in punched card form for subsequent processing; and 4) the ability to format and output the results of a query on magnetic tape or printer hardcopy. To satisfy these requirements it was decided to adapt an operational generalized information system to the specifications of the Library of Congress. The system chosen was AEGIS, designed and implemented by Programmatics, Inc. The modification is known as the MARC Retriever.

General Description

The MARC Retriever comprises four parts: a control program, a parser, a retrieval program, and a utility program. Queries are input in the form of punched cards, stacked in the core of the IBM S/360, and operated on as though all queries were in fact one query. Thus a MARC record will be searched for the conditions described by all queries, not by handling each query individually and rewinding the input tape before the next query is processed.

The control program is the executive module of the system. It loads the parser and reads the first query statement. The parser is then activated to process the query statement. On return from the parser, the control program either outputs a diagnostic message for an erroneous query or assigns an identification number to a valid query. After the last query statement has been parsed, the control program loads the retrieval program and the MARC input tape is opened. As each record on the MARC tape is processed, the control program checks for a valid input query. If the query is valid, the control program branches to the retrieval program. On return from the retrieval program, the control program writes the record on an output tape if the record meets the specifications of the query. After the last MARC record has been read from the input tape, the control program branches to the retrieval program for final processing of any requested statistical function (HITS, RATIO, SUM, AVG) that might be a part of the query. The output tapes are closed and the job is ended.

The parser examines each query to insure that it conforms to the rules for query construction. If the query is not valid, an error message is returned to the control program giving an indication as to the nature of the error. Valid query statements are parsed and converted to query strings in Polish notation, which permits mathematical expressions without parentheses. The absence of embedded parentheses allows simpler compiler interpretation, translations, and execution of results.

The retrieval program processes the query strings by comparing them with the MARC record data elements and the results of the comparison are placed in a true/false stack table. If the comparison result is true, output is generated for further processing. If the result is false, no action takes place. If query expressions are linked together with "OR" or "AND" connectors, the results in the true/false stack table are ORed and ANDed together resulting in a single true or false condition.

The utility program counts every data element (fixed field, tag, indicator, subfield code, data in a variable field) that is used in a query statement. The elements in the search argument are counted separately from those in the output specifications. After each run of the MARC Retriever, the counts can be printed or punched for immediate use, or they can be accumulated over a longer period and processed on demand.

Query Language

General.

Query statements for the MARC Retriever must be constructed according to a precisely defined set of rules, called the syntax of the language. The language permits the formation of queries that can address any portion of the MARC record (fixed fields, record directory, variable fields and associated indicators and subfields). Queries are constructed by combining a number of elements: MARC Retriever terms, operators, fixed field names, and strings of characters (hereafter called constants). The following sections describe the rules for constructing a query and the query elements with examples of their use.

Query Formation.

A query is made up of two basic parts or modes: the if mode which specifies the criteria for selecting a record; and the list mode which specifies which data elements in the record that satisfy the search criteria are to be selected for printing or further processing. In general, the rules that apply to constructing if-mode expressions apply to constructing list-mode expressions except that the elements in the list mode must be separated by a comma. A generalized query has the following form:

IF if-mode expression LIST list-mode expression;

Where:

IF	Signals the beginning of the if mode.
if-mode expression	Specifies the search argument.
LIST	Signals the beginning of the list mode.
list-mode expression	Specifies the MARC record data element(s) that are to be listed when the search argument specified in the if-mode expression is satisfied.

The format of the query card is flexible. Columns 1 through 72 contain the query which may be continued on subsequent cards. No continuation indicator is required. Columns 73 through 80 may be used to identify the query if desired. The punctuation rules are relatively simple. One or more blanks must be used to separate the elements of a query and a query must be terminated by a semicolon.

Queries that involve fixed fields take the following form:
IF fixed-field-name1 = constant LIST fixed-field-name2

Where:

fixed-field-name1	The name of fixed field.
=	Any operator appropriate for this query.
constant	The search argument
fixed-field-name2	The fixed field to be output if a match occurs.

To query or specify the output of a variable field, the following general expression is used.

IF SCAN (tag=nnn) = constant LIST SCAN (tag=nnn);

Where:

SCAN	Indicates that a variable field is to be referenced.
tag	Indicates that the tag of a variable field is to follow.
=	The only valid operator.
nnn	Specifies the tag of the variable field that is to be searched or output.
constant	Specifies the character string of data that is the search argument.

The MARC Retriever processes each query in the following manner. Each record in the data base is read from tape into core and the data elements in the MARC Record specified in the if-mode expression are compared against the constant(s) in the if-mode expression. If there is a match, the data element(s) specified in the list-mode expression are output.

Key Terms.

The terms used in a query statement fall into two classes. The first group instructs the program to perform specified functions: SCAN, HITS, AVG, RATIO, SUM. The second group relates to elements of the record structure. The most important key terms in this class are: INDIC (indicator), NTC (subfield code), RECORD (the entire bibliographic record), and TAG (variable field tag). These terms are used to define a constant; e.g., TAG = 100.

Operators.

Operators are characters that have a specific meaning in the query language. They fall into two classes. The first contains relational operators, such as equal to and greater than, indicating that a numeric relationship must exist between the data element in the MARC record and the search argument. The second class comprises the logical operators "and" and "or". The operators of the MARC Retriever are shown in Table 2. In the definitions, C is the query constant and D is the contents of a MARC record data element.

Table 2. Operators of the MARC Retriever

Operator	*Meaning*
=	C equals D
>	C is greater than D
⩾	C is greater than or equal to D
<	C is less than D
⩽	C is less than or equal to D
¬=	C is not equal to D
&	"and" (both conditions must be true)
|	"or" (at least one condition must be true)

Constants.

A constant is either a string of characters representing data itself (e.g., Poe, Edgar Allan) or a specific variable field tag, indicator(s), and subfield code(s). Constants may take the following form:

CC	Where CC is an alphabetic or numeric character or the pound sign "#". When this form is used, the MARC Retriever will convert all lower-case alphabetic characters in the data element of the MARC record being searched to upper-case before a comparison is made with search argument. This conversion feature permits the use of a standard keypunch that has no lower-case capability for preparation of queries.
'CC'	Where CC can be any one of the 256 characters represented by the hexadecimal numbers 00 to FF. This form allows nonalphabetic or nonnumeric characters not represented on the standard keyboard to be part of the search argument. When this form is used, the MARC Retriever will also convert all lower-case alphabetic characters in the data elements in the MARC record being searched to upper-case before a comparison is made.
@CC@	Where CC can be any one of the 256 characters represented by the hexadecimal numbers 00 to FF. When this form is used, characters in the data element of the MARC record being searched will be left intact and the search argument must contain identical characters before a match can occur.
#	The pound sign indicates that the character in the position it occupies in the constant is not to take part in the comparison. For example, if the constant were #ANK, TANK, RANK, BANK would be considered matches. More than one pound sign can be used in a constant and in any position.

Specimen Queries.

The following examples illustrate simple query statements involving fixed and variable fields.

IF MCPDATE1 = 1967 LIST MCRCNUMB;

The entire MARC data base would be searched a record at a time for records that contained 1967 in the first publication date field (MCPDATE1). The LC card number (MCRCNUMB) of the records that satisfied the search argument would be output.

IF SCAN(TAG = 100) = DESTOUCHES LIST SCAN(TAG = 245);

The personal name main entry field (tag 100) of each MARC record would be searched for the surname Destouches. If the record meets this search argument, the title statement (tag 245) would be output.

In addition to specifying that a variable field is to be searched, the SCAN function also indicates that all characters of the variable field are to be compared and a match will result at any point in the variable field where the search argument matches the variable field contents. For example, if the if-mode expression is SCAN(TAG = 100) = SMITH a match would occur on the following examples of personal name main entries (tag 100): SMITH, JOHN; SMITHFIELD, JEROME; JONES-SMITH, ANTHONY.

It is possible to include the indicators associated with a variable field in the search by augmenting the constant of the SCAN function as follows:

IF SCAN(TAG = 100&INDIC = 10) = DESTOUCHES
LIST SCAN(TAG = 245);

Where:

INDIC	Specifies that indicators are to be included.
1	Specifies that the first indicator must be set to 1 (the name in the personal name main entry [tag 100] is a single surname,
0	Specifies that the second indicator must be set to zero (main entry is not the subject).

The personal name main entry field (tag 100) of each record would be searched and a hit would occur if the indicators associated with the field were 1 and 0 and the contents of the field contained the characters "Destouches." If the record met these search criteria, the title statement (tag 245) would be output. It is also possible to restrict the search to the contents of one or more subfields of a variable field.

For example:

IF SCAN(TAG = 100&INDIC = 10&NTC = A) = DESTOUCHES
LIST SCAN(TAG = 245);

Where:

NTC	Indicates that a subfield code follows.
A	Specifies that only the contents of subfield A are to be included in the search. Note that in this form the actual subfield code "a" is converted to "A" by the program (see section on Constants).

Special Rules.

So far the discussion has concerned rules of the query language that apply to either the if mode or the list mode. This section and the remaining sections will discuss those rules and functions that are unique to either the if mode or the list mode.

In the if mode, fixed and variable field expressions can be ANDed or ORed together using the logical operators & and |. For example:

```
IF MCPDATE1 = 1967&SCAN(TAG = 100) = DESTOUCHES
                LIST SCAN(TAG = 245);
```

This query would search for records with a publication date field (MCPDATE1) containing 1967 and a personal name main entry field (tag 100) containing Destouches. If both search criteria are met, the title statement field (tag 245) would be printed.

In the list mode more than one fixed or variable field can be listed by a query as long as the fixed field names or scan expressions are separated by commas. For example:

```
IF SCAN(TAG = 100) = DESTOUCHES
LIST SCAN(TAG = 245), MCRCNUMB;
```

The list mode offers two options, LIST and LISTM, which result in different actions. LIST indicates that the data elements in the expressions are to be printed, and LISTM indicates that the data elements in the expression are to be written on magnetic tape in the MARC processing format.

It is often desirable to list a complete record either in the MARC processing format using LISTM or in printed form using LIST. In either case, the listing of a complete record is activated by the MARC Retriever key term RECORD. For example:

```
IF SCAN(TAG = 100) = DESTOUCHES LIST RECORD;
```

The complete record would be written on magnetic tape in the MARC processing format instead of being printed out if LISTM were substituted for LIST in the above query.

Four functions can be specified by the LIST mode. HITS signals the MARC Retriever to count and print the number of records that meet the search criteria. For example:

```
IF SCAN(TAG = 650) = AUTOMATION LIST HITS;
```

RATIO signals the MARC Retriever to count both the number of records that meet the search criteria and the number of records in the data base and print both counts.

The remaining two LIST functions permit the summing of the contents of fixed fields containing binary numbers. SUM causes the contents of all specified fields in the records meeting the search criteria to be summed and printed. For example:

```
IF MCRCNUMB = 'b̸b̸b̸68 ######' LIST SUM (MCRLGTH);
```

The data base would be searched for records with LC card number field

(MCRCNUMB) containing three blanks and 68 in positions one through five. The remaining positions would not take part in the query process and could have any value. If a record satisfied this search argument, the contents of the record length field (MCRLGTH) would be added to a counter. When the complete data base had been searched, the count would be printed. AVG performs the same function as SUM and also accumulates and prints a count of the number of records meeting the search criteria.

Use of the Program

The MARC Retriever has been operational at the Library of Congress since May 1969 and selected staff members representing a cross-section of LC activities have been trained in the rules of query construction. The applications of the program to the MARC master file include: identification of records with unusual characteristics for the format recognition study; selection of titles for special reference collections; and verification of the consistency of the MARC editorial process. As the file grows, it is expected that the MARC Retriever will be useful in compiling various kinds of bibliographic listings, such as translations into English, topical bibliographies, etc., as well as in making complex subject searches.

The MARC Retriever is not limited to use with the MARC master file; it can query any data base that contains records in the MARC processing format. Thus, the Legislative Reference Service is able to query its own data base of bibliographic citations to produce various outputs of use to its staff and members of Congress.

Because the MARC Retriever is designed to conduct searches from magnetic tape, it will eventually become too costly in terms of machine processing time to operate. It is difficult to predict when the system will be outgrown, however, because its life span will be determined by the growth of the file and the complexity of the queries. Meanwhile, the MARC Retriever should provide the means for testing the flexibility of the MARC format for machine searching of a bibliographic file.

REFERENCES

1. U.S. Library of Congress. Information Systems Office: *The MARC Pilot Project.* (Washington, D.C.: 1968), pp. 40-51.
2. Rather, John C.; Pennington, Jerry G.: "The MARC Sort Program," *Journal of Library Automation,* 2 (September 1969), 125-138.
3. RECON Working Task Force. *Conversion of Retrospective Catalog Records to Machine-Readable Form.* (Washington, D.C.: Library of Congress, 1969).
4. U.S. Library of Congress. Information Systems Office: *Subscribers Guide to the MARC Distribution Service,* 3d ed. (Washington, D.C.: 1969), pp. 31-31b.
5. Ibid., p. 40.

6. Cunningham, Jay L.; Schieber, William D.; Shoffner, Ralph M.: *A Study of the Organization and Search of Bibliographic Holdings Records in On-Line Computer Systems: Phase I.* (Berkeley, Calif.: Institute of Library Research, University of California, 1969), pp. 85-94.
7. Jolliffe, John: "The Tactics of Converting a Catalogue to Machine-Readable Form," *Journal of Documentation*, 24 (September 1968), 149-158.