James L. Ames
Information Systems Management Corp.
IBM Branch Office
Pullman, Washington

PL/I FOR LIBRARY SYSTEMS

I would like to begin by emphasizing that what we are considering is a system—in particular, a computer-aided library system. The system in this sense may be a technical services system, a serial records system, or a library holdings information retrieval system. Each of these library systems, when implemented, will consist of a series of interrelated computer programs which operate on a store of information, or a data base. These programs either keep the contained information current to reflect the true status of the system, or provide reports, documents, and current information to the users of the system.

Certainly these concepts, updating and querying a data base, are not new to the data processing community. These concepts, in fact, exist in every data processing organization at various levels of sophistication. A simple example of such information processing might be found in a payroll system involving a file of employee information, a data base, and a set of computer programs to process the data. The data base in this case is updated by the addition of new employees, by salary or benefit changes, by deletion of former employees, or possibly by accumulation of year-to-date totals during processing. The products of the current file would be the pay checks for the employees and the reports used by the company and the company management.

At the other end of the scale, one might find a corporate information system. In part of its data base might be all of the information that affects the budget of the corporation. Such data, which come from widely varied sources, must be constantly updated to reflect the current status of the budget. The data must be available for regular processing and for the management of the corporation to retrieve and correlate as desired. Management will then be able to "converse" with the computer to obtain information stored or developed from information in the data base and thus have the capability to explore new avenues as new ideas develop.

This latter capability is very desirable in the library as well as the corporation. An example of this approach in the library is found in providing an untrained library user with the capability to query the holdings of a library by using author, title, or subject as entries. The data base in this case consists of the library catalog. The user, who has a predefined concept relating to the information for which he is searching, should have the capability to direct the search and explore different avenues that may lead to the end result, a probable source of information.

These types of systems are constructed by using one or another kind of programming language. The language may be an assembly level language, whose instructions closely resemble those machine functions that are built into the computer, or a procedural language such as FORTRAN, COBOL, ALGOL, or PL/I. Each statement in a procedural language normally generates several machine-level instructions.

The programming language requirements of the library system, as compared to that of the corporate information system are very similar. For instance, in both systems we must have the capability to manipulate large files in both a random and a sequential manner. In both systems we must be able to do arithmetic, and have flexible facilities to print reports. If the system is an on-line system, it is necessary to have the ability to intercept errors and interrupts so that, for instance, program failure caused by one terminal in the system does not affect the operation of the other terminals using the same program. In addition, for certain errors, the program can analyze and feed back to the user the nature of the error that caused program failure.

The library system, as one may expect, has all the requirements of the corporate system as well as some additional ones. These additional requirements are caused mainly by the types of data that are encountered in the library. Outside the library, one tends to find that most information in a computer record is defined with a fixed length. That is, information such as dollars, stock numbers, and often names have a fixed length field set up both within the computer and on external storage. Typically, one might find defined in a card, "name" in columns 1 through 20, "stock number" in columns 29 through 30, and "price" in columns 31 through 39. Data items that do not vary greatly in length are placed into fixed length fields, and for a very good reason: it is much easier to write programs and manage the data using fixed fields.

The problems of handling data become much more acute when working with library information such as is found is bibliographic records. Authors, titles, collations, and subjects are of variable and unpredictable length. Since a large portion of library programming deals with such fields, it is imperative to have the proper tools built into a programming language to manipulate easily this type of information. It is equally important to have these tools so built in that the programmer can conceptualize the data as it really is. For example, to the programmer a title may be a string of characters call "title," or it may be an array of numbers that must be handled with arithmetic type statements and loops. If the programmer can work in terms of characters—that is, search for character patterns, compare character strings, extract strings of characters, insert strings of characters, and concatenate strings of characters, rather than

manipulating arrays of numbers—he will be more productive and the programs that he writes will be more logical and understandable.

What kind of capabilities should one look for in a programming language for a library system? Perhaps a representative example would help to point out which capabilities are necessary and which may be desirable. I would like to use as this example an experimental system which I developed while at Washington State University in 1968. The system was developed as a tool to test techniques for retrieving titles from a computer-based file using an imperfect search request. The research was directed towards providing a library user with the ability to query the holdings of the library. The system, I think, provides most of the facilities necessary in a programming language when it is used to develop library systems.

For a user of this system, the question is: "Is this title held by the library?" After the query is entered, it is analyzed by the system which takes into account possible spelling errors, missing or additional words or phrases, and other mistakes such as capitalization and punctuation. Based upon the analysis, the system will search the catalog and report the findings of the search to the user. The results of the search are either a list of "most probable" matches to the query, or a negative reply. The user can then accept the results or rephrase the query. An example of the query and the results is found in Figure 1.

The system consists of two main processes which I would like to examine in some detail to point out the different aspects of the system as far as programming is concerned. The first of these processes constructs the library catalog on a direct access device. The second analyzes search requests, searches the catalog, and reports the results of the search to the system user.

The catalog together with its indexes, is built from examining the title in raw bibliographic form. The input to the catalog building phase is a machine-readable bibliographic record such as distributed by the MARC Project. The files used by the system consist of two primary components, the dictionary and the catalog. The general organization of the files within the system may be seen in Figure 2. The catalog contains the complete bibliographic citations and is used to display search results to the user. The dictionary component is used interactively to formulate the search request and for the primary phase of the search strategy. Each component of the system is assumed to have entries available by random access techniques. Note that in the case of the dictionary entry, a direct access address in this system must be found by manipulating a string of characters. In addition to manipulating the word for the dictionary entry, the programming language must have the facility to store and retrieve both fixed length and variable length records on a direct access device.

The dictionary in Figure 2 contains all the non-common words that appear in the bibliographic entries. These words have been analyzed and modified to compensate for spelling and typographical errors. Along with each word entry in the dictionary is additional information that describes the context that the word is used in, "see" references, "see also" references, and a list of documents containing the word in the given context. The context or attribute entry describes the type of entry in which the word was used, such

Figure 1

QUERY AND SYSTEM RESPONSE EXAMPLES

QUERY: TITLE: India's Foreign Policies.

30867 0.8667 India's Defense and Foreign Policies.

END

QUERY: TITLE: MIT TECHNICAL REPORT

42136 1.0000 M.I.T. Technical Reports.

END

QUERY: TITLE: Les Integrals Eulerines.

01862 1.0000 Les Integrales Eulerinnes et Leurs Applications.

END

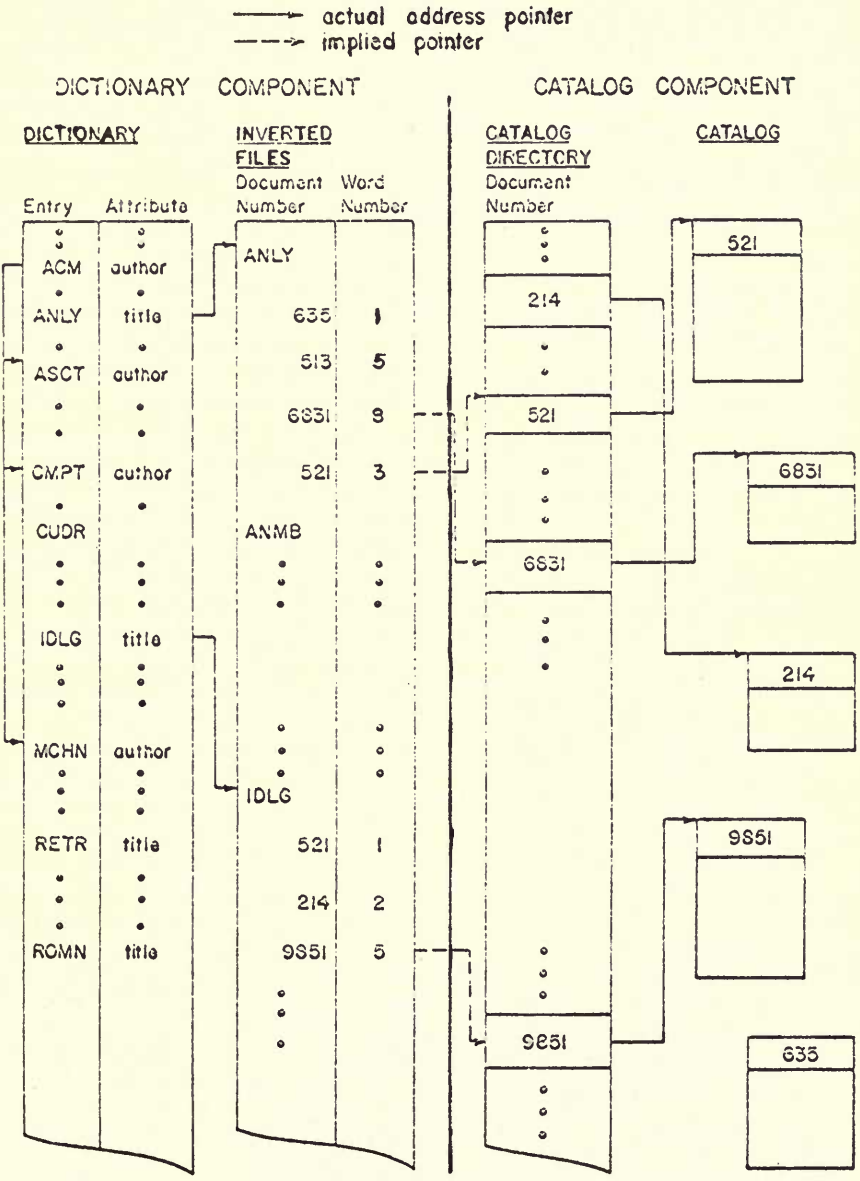
QUERY: TITLE: Websters Seventh new Collegiate Dictionary.

THIS TITLE NOT HELD BY THE LIBRARY.

Note: In the above examples the first number in the system response is the accession number and the second is the coefficient of similarity between the query title and the catalog title as calculated by the system.

Figure 2

Catalog Searching System Organization



as a title entry or an author entry. The "see" reference points to a word to be used instead of the one in the search request. The "see also" reference points to the associated words to be used in addition to the ones appearing in the search request. Both of these cross reference facilities are used to expand or standardize abbreviations, and to aid the user in formulating his search request. Each entry in the document list contains two fields. One indicates a document number that the word was used in, and the other the relative position of this word within the entry. The relative position within the entry is used in the algorithm to measure the degree of similarity between the search request and a title in the catalog.

The catalog component of the system provides for storage of the catalog and makes each catalog record directly accessible. The catalog directory as shown in Figure 2, provides a list of catalog records together with their physical location in the file. The program must be able to manipulate the document identifier to yield a mapping from the identifier to the catalog directory. It must be able to manipulate the variable length catalog record either as a variable length record or as a series of fixed length records with a variable number of records.

The dictionary is built by extracting a title from the bibliographic record, manipulating, and analyzing it. This title is processed as follows:

1) All punctuation is removed from the title by serially examining each character. Some are changed to blanks; others are deleted.

2) Each lower case alphabetic character is changed to upper case by logically OR'ing it with a bit string of 01000000 on the IBM 360.

3) The title is broken down into its individual words and the relative position of these words is marked. This is accomplished by searching for the blanks between words.

4) Each word is then processed against a list of common words, and the common words are deleted from the list of title words.

5) Each remaining word is examined character by character from left to right and the abbreviated form of the word is constructed. This abbreviated form is especially constructed to compensate for spelling variations.

6) Each remaining triple (word, document number, and relative position) is then added to the dictionary.

For an example, see Figure 3. Note that the above procedure involves considerable character string searching, character string comparing, substring extraction, character manipulation, character string concatenation, character string comparisons, and logical operations. Adding to the dictionary again requires the ability to read and update the direct access files.

When a query to the file is made, the search request title is processed as in steps 1-5 above. The dictionary is then searched for each of the remaining words. The list of document number–word position pairs–is retrieved for each non-common word found in the dictionary. Each list is sorted by document number as the primary sort key and secondarily by word position. Each document number in each list is then associated with its corresponding word, and the lists are merged to form as an end result a reconstruction of the original titles as they appeared after being processed, as in steps 1-5 above.

Figure 3

AN EXAMPLE OF TITLE AND QUERY PREPROCESSING PRIOR TO MATCHING.

STEP	RESULTS	
0	Design Principles for an On-Line Information Retrieval System.	
1	Design Principles for an On-Line Information Retrieval System.	
2	DESIGN PRINCIPLES FOR AN ONLINE INFORMATION RETRIEVAL SYSTEM.	
3	DESIGN	1
	PRINCIPLES	2
	FOR	3
	AN	4
	ONLINE	5
	INFORMATION	6
	RETRIEVAL	7
	SYSTEM	8
4	DESIGN	1
	PRINCIPLES	2
	ONLINE	5
	INFORMATION	6
	RETRIEVAL	7
	SYSTEM	8
5	DSGN	1
	PRNC	2
	ONLN	5
	INFR	6
	RTRV	7
	SYST	8

The search request list is paired with each title list successively and input to a match/comparison routine. The match routine computes a coefficient of similarity between the search request and the title. The coefficient is a function of the number of matching words, the order of the matching words and the relative positions of the words within the titles. Based upon the magnitude of the similarity coefficient, the title is either rejected or added to the list of titles to be considered further.

Phase two of the search uses that list of titles to be further considered. A comparison is made among these titles to find the "best" match to the search request. The title to be compared to phase two is extracted from the bibliographic record in the catalog and is processed as before except that non-common words are not deleted.

The preceding system was successfully programmed using PL/I level F in a period of less than two months. This short period of time for a system of this complexity and diversity of demands upon the resources of a computer is attributable to PL/I for several reasons expanded upon below.

Many different programming techniques and operations were necessary to accomplish this task. Very involved character manipulation was needed. The ability to build and access several direct access files using different accessing techniques was needed. The ability to handle variable length records and variable length data was necessary. An arithmetic capability was needed to calculate similarity coefficients, calculate direct access addresses, and provide various counters. It was necessary to perform both internal and external sorts. PL/I provided the ability to perform each of these in a direct and easily programmed manner. The internal sort was programmed in PL/I in about half a day.

A second important aspect of the language that aided in the construction of the system was the ability to construct and test small components of the system independently. The system was constructed in a highly modular manner which, in addition to de-bugging, provided the ability to modify individual components of the system. For instance, the matching algorithm could be easily changed without affecting the rest of the system. PL/I as a language lends itself very readily to modular program construction.

The speed with which the system was developed was accelerated significantly by the excellent diagnostic capability of the language. The IBM PL/I implementation provides extensive diagnostic messages and analysis both at the time that the program is compiled, and when the program is executing. The execution time error handler provides in addition, the ability for the programmer to diagnose and recover from likely errors through the use of the ON statement.

In addition to the capabilities mentioned above, PL/I provides very flexible format control for printing reports, catalogs, or catalog cards if so desired. In addition, the same flexible format control is available to build strings of text internally. Also of significance to the library or text-handling user is the list-processing feature of the language. This feature provides a very flexible and efficient manner to handle text and lists of unknown length.

To utilize the flexibility and scope of the language, one must pay a price. This price is probably realized in terms of slightly larger programs and slower compilation speed. In my opinion, however, this price is very small when weighed against the productivity gain realized by the library programmer and the advantages of having a single comprehensive language to learn and use when implementing a library system.