Management Services: A Magazine of Planning, Systems, and Controls

Volume 3 | Number 3 Article 7

5-1966

Problem-Oriented Languages: FORTRAN vs. COBOL

Wayne S. Boutell

Follow this and additional works at: https://egrove.olemiss.edu/mgmtservices

 Part of the Accounting Commons

Recommended Citation

Boutell, Wayne S. (1966) "Problem-Oriented Languages: FORTRAN vs. COBOL," *Management Services: A Magazine of Planning, Systems, and Controls*: Vol. 3: No. 3, Article 7.
Available at: https://egrove.olemiss.edu/mgmtservices/vol3/iss3/7

This Article is brought to you for free and open access by eGrove. It has been accepted for inclusion in Management Services: A Magazine of Planning, Systems, and Controls by an authorized editor of eGrove. For more information, please contact egrove@olemiss.edu.

"Software" is rapidly becoming the most pressing problem in the data processing world. This article discusses the pros and cons of two of the most common of the new—

PROBLEM-ORIENTED LANGUAGES: FORTRAN VS. COBOL

by Wayne S. Boutell
University of California

ONE OF THE MOST significant developments in the computer revolution during the last five years has been the use of problem-oriented languages. These languages have opened the door to vastly increased opportunities for utilization of the computer, and they have greatly reduced the programing effort required to accomplish desired objectives.

Since most businessmen have never had a course in computer programing and since the computer is becoming increasingly important in business applications, it has become mandatory for accountants, production managers, operations researchers, and other personnel in business firms to acquire some understanding of the potential of these problem-oriented languages. This understanding should further an appreciation of the problems faced by data processing personnel and promote more satisfactory communication between the technical experts and the business managers at all levels of the business firms. Because of the relative simplicity of problem-oriented languages, it is possible to obtain a grasp of the basic concepts of these languages in fifteen to twenty hours of study.

A major reason for the use of problem-oriented languages is the increasing shortage of qualified programers at all levels. Since these languages can be mastered within a relatively short span of time, programers can be trained more readily. The use of problem-oriented languages not only facilitates training but also reduces the program-

Published by eGrove, 1966

ing effort necessary to obtain acceptable programs by as much as 80 per cent. The expert programers claim that this gain is at least partially offset by a certain sacrifice in the efficiency of the machine language program, and this point would certainly have to be granted; but, on balance, the bulk of the evidence indicates that eventually problem-oriented languages will replace the so-called machine-oriented languages both in business data processing applications and in scientific research.

It has become almost mandatory for business managers to acquire a working knowledge of one of the problem languages. Fortunately, they are so simple that a grasp of basic concepts can be gained in fifteen to twenty hours of study.

Problem-oriented languages

What is a problem-oriented language? In essence, it is a language that approximates the language of the problem that is to be solved by the computer. Since the language of the problem bears little resemblance to the instructions required by the computer to process the data, it is necessary to provide the computer with a translator that will convert the problem-oriented language to machine-oriented language that can be interpreted by the computer circuitry. This translator is itself a computer program, and it is generally referred to as a compiler.

Each computer manufacturer has the responsibility of providing these compiler programs, thus freeing the programer from many conventions and restrictions inherent in machine-oriented languages. The compiler, in addition to the translating function, also scans the problem-oriented language program for syntactical errors and provides error messages (sometimes called diagnostics) to the programer to ease the debugging of the program.

It should be noted that a more recent development is the introduction of PL/1 (Programing Language — First Version) for the IBM System/360 line of computers. PL/1 is a problem-oriented language that incorporates the basic features of COBOL, FORTRAN, and ALGOL within a single schema. It is modular in character. A relatively inexperienced user can learn that subset of the language which is suited to his particular programing requirements, yet the language is sufficiently general so that it is useful to any computer user. In view of the fact that PL/1 is not yet fully developed, the subsequent discussion will be limited to those languages that are in use at the present time.

The principal problem-oriented languages that are in general use today are illustrated in the following diagram:

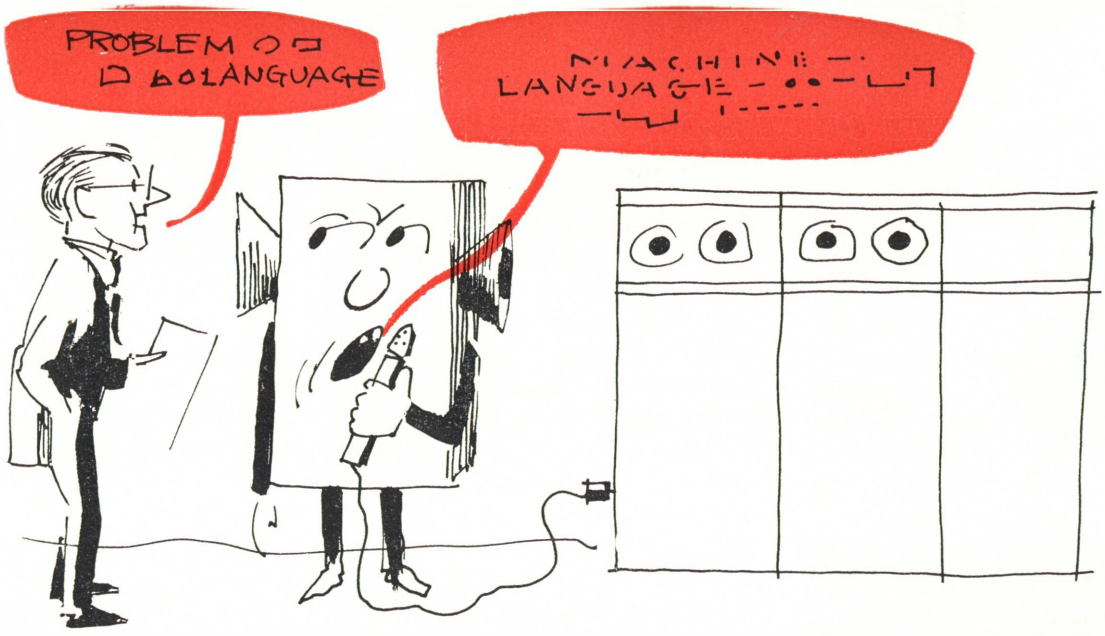
Basic Language	Corresponding Problem-oriented Language
Verbalization as represented by the English Language	COBOL
Quantification as represented by mathematical notation	FORTRAN, ALGOL

Characteristics of COBOL

If the problem, such as a business transaction, can be stated in ordinary English, then the most suitable problem-oriented language is probably COBOL. COBOL is an abbreviation for "COmmon Business Oriented Language." This language was developed in 1959 by a committee composed of government users and computer manufacturers.[1] The COBOL-61 version of this language is the one that is most widely used today although various manufacturers have made changes since COBOL-61 was adopted. For example, IBM 7094 COBOL is in its thirteenth version.

Nature of FORTRAN

FORTRAN is the oldest of the problem-oriented languages, having been developed originally in 1956 by IBM for use with the IBM 704 computer. FORTRAN is an abbreviation for "FORmula TRANslation" and has been primarily used for scientific and mathematical programing problems. However, there is an increasing conviction that FORTRAN may also be adaptable to business problems, and it has



Before the development of machine translators, communication was difficult between the problem language of the programer and the machine language of the data processor.

been used successfully in a number of situations.[2]

Characteristics of ALGOL

The problem-oriented language that is closest to the symbolism employed in mathematical and statistical notation is ALGOL. ALGOL, which stands for ALGOrithmic Language, is exactly what its name implies: a language for expressing algorithms.[3] ALGOL was developed originally to standardize communication among mathematicians of the world and to provide a basis for common understanding and a precision of exposition.

Best business languages

Since this article is primarily concerned with the potential of problem-oriented languages in business-type problems, the remainder of the discussion will center around the various aspects of FORTRAN and COBOL. ALGOL appears to be primarily suited to problems that either do not exist in business situations or can be expressed in FORTRAN.

The IBM System/360 provides the facility for compiling programs written in either FORTRAN or COBOL. Apparently, IBM feels that the announced purpose of the System/360, that of meeting the requirements of scientifically oriented users as well as business-oriented users within the framework of one set of hardware components, can be accomplished by utilizing these two problem-oriented languages. This development, of course, does not imply any commitment as to which language would be used for which type of application. It is also interesting to note that IBM has apparently de-emphasized the symbolic-type intermediate languages that are used in business today.[4] The Burroughs B-5000 also de-emphasizes the use of machine-oriented languages.

WAYNE S. BOUTELL, Ph.D., CPA, is the associate dean of the School of Business Administration and associate professor of accounting at the University of California, Berkeley. Mr. Boutell is a member of the American Institute of Certified Public Accountants and the American Accounting Association. He has contributed two books to the field of accounting: Auditing With the Computer and Fortran and Business Data Processing. In addition, he has contributed articles to The Journal of Accountancy and The Accounting Review.

Pros and cons

Before examining the differences between FORTRAN and COBOL, it is well to review certain basic characteristics of a problem-oriented language. As mentioned earlier, the use of a compiler (translator) is essential to the implementation of any problem-oriented language. Therefore, the efficiency of any problem-oriented language depends in large measure on the efficiency of the compiler that was written by the computer manufacturer to translate the problem-oriented language. Understandably, the efficiency of compiler programs varies among the several computer manufacturers. Therefore, the comparison between two languages such as FORTRAN and COBOL is a measure both of the language itself and of the efficiency of the compiler. These two aspects of the problem are not separable. Therefore, conflicting conclusions may be

Published by eGrove, 1966

THE ORIGINS OF "AUTOMATIC PROGRAMING"

One of the reasons for the growing use of computers by industry—and perhaps the most important reason for their widespread use today—is the simplifications that have been made in the task of programing the machines. Basically a computer recognizes only two impulses, on and off, which can represent the symbols 1 and 0 used in binary notation. By use of the two symbols and a binary numbering system, any decimal number can be represented, as well as a number of alphabetic words. (See "Electronic Data Processor," M/S, March-April, '64.)

This series of binary numbers is the machine language, the code which the computer can work with.

When computers were first introduced it was necessary for human programers to translate each of the steps in a carefully designed computer program into its computer code equivalent in binary numbers. Thus, the instruction "load address" might have been written as 01000001.

The first step toward what is sometimes euphemistically called "automatic programing" came with the development of an assembly program which the machine could use to translate symbols or Arabic letters into their binary equivalents. In effect, the machine was helping create its own programs by translating each command into the necessary binary number and punching it on a card. These cards, produced by the computer, could then be fed back into the computer with cards containing the raw data of the problem to be solved. "Load address" could now be LA instead of 01000001 as far as the human programer was concerned.

Further refinements have all stemmed from this first great simplification. They have in effect been further developments of the computer's ability to help in the simple clerical task of creating its own programs in all their detail as long as the actual operations that must be performed are first spelled out for the machine by the human programer. The next step after the development of the first rudimentary assembly programs was allowing the computer to assign locations within its memory for instructions that had to be stored.

So far, writing a program, though infinitely simpler than it had been in the days when the program had to be prepared in the machine's language, was still complicated by the fact that the computer required one written instruction for each of its operations. With the development of macro instructions this problem too was greatly eased. Every computer program has certain common instructions which are repeated several times during the course of the program. These instructions — read a card or tape, store for print — although simple in English, require detailed instructions to the computer as to just what actions must be taken. Through macro instructions, the detailed sequential steps the computer must take to execute one of these simple commands were programed into the computer itself, so that one simple command in symbolic language could trigger a whole series of machine-language instructions for the computer.

It was as though instructions were given first — after the phase when the human programer had to put every instruction in machine language — in a type of pidgin English where every command had to be spelled out in detail. Then as programers grew more skilled and machines more sophisticated, a higher type of language evolved in which the machine performed a whole series of actions — or rather programmed itself to perform a whole series of actions — on the basis of one instruction.

A very rough and imperfect analogy would lie in the instructions given a small child asked to do something for the first time. At first they would be extremely detailed, telling him exactly what to do and how to do it. As the child grew older, though, the detailed instructions could contract to one sentence. The difference with the computer was that detailed machine instructions were still necessary, but the machine could produce them itself, given the one overriding instruction.

The effect of macro instructions, of course, was tremendous. Programers could be trained more quickly and easily, and, most important of all, the time required to write programs was reduced. Since even today the short supply of programers represents a bad bottleneck in electronic data processing, it can be imagined how much more critical this would be if it took as long to train programers as it once did, and as long to write a program.

In the time interval since the introduction of macro instructions, programing languages have grown immensely more sophisticated and versatile. Pidgin English has evolved into a fairly respectable form of Basic English, with its own rigid rules of grammar and syntax.

The two programing languages discussed in this article, FORTRAN and COBOL, were developed, respectively, by IBM and the Department of Defense. COBOL is a language acceptable to all major computers; it was developed as a common language. But many manufacturers have their own specialized languages as well, which in most cases are more efficient on their machines than COBOL.

reached by two people carrying out the comparison on different equipment, utilizing different compiler programs.

Advantages and disadvantages

However, although these differences may be difficult to quantify, it is certain that, in general, the use of a problem-oriented language inherently contains the following advantages:

Reduces programer training cycle

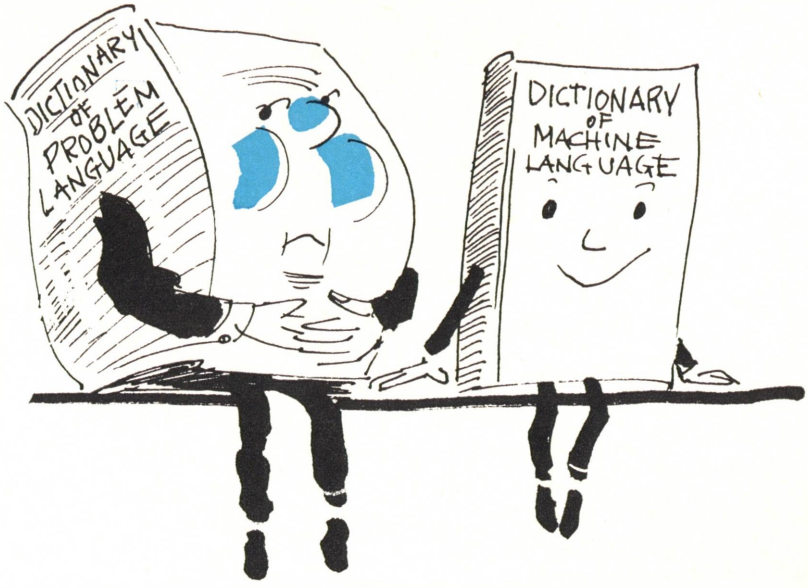
Reduces programing effort to accomplish a given objective

Provides an instructional medium that de-emphasizes technical rules.

It is certainly true that the use of problem-oriented languages contains certain concomitant disadvantages also:

Resulting machine-language programs are relatively inefficient as compared to symbolic or absolute machine-language coding.

The programs written in problem-oriented languages are difficult to relate to the computer hardware and may complicate debugging.



The language of the problem to be solved has little resemblance to the numerical language of the computer which must solve it.

FORTRAN

What is the basic structure of FORTRAN? FORTRAN is composed of a relatively highly formalized set of rules primarily adaptable to statements that approach mathematical formulations. But it is also true that many business transactions are reducible to precise mathematical expressions. For example, the computational algorithm normally used in connection with payroll can be quantified rather readily. The basic structure of the computation (in the case of hourly payroll) involves a multiplication of an hourly rate by a weekly total of hours in order to obtain a gross pay amount. Additional complications may be visualized, such as overtime pay or night shift differential, which might complicate the algorithm.

Let us assume that the base pay rate is $1.50 per hour. Let us further assume that the employee worked 42 hours during the week, that there was no night shift differential involved in the calculation, and, finally, that overtime is paid for hours worked in excess of 40 hours. Our manual calculation might be somewhat as follows:

$$\begin{array}{rrr} 1.50 & \times & 42 = 63.00 \\ 0.75 & \times & 2 = 1.50 \\ \hline \text{Gross Pay} & & 64.50 \end{array}$$

If we were to present a similar statement as a part of a FORTRAN computer program, it might look something like this:

```
GROSS = (RATE * HOURS) +
(HOURS - 40.0) * RATE/2.0
```

Arithmetical operations can be precisely stated in FORTRAN in a completely general way. The primary difficulty with FORTRAN is in the more complicated programs where the significance of the arithmetic statements is not readily determinable from the context. There may be need for interpretative comments to accompany the FORTRAN program. Additional complications may develop because of the relative formality of the input/output instructions and the restrictiveness of quantifying precisely certain business transactions.

COBOL

What is COBOL? COBOL is a highly structured subset of English. As already mentioned, COBOL is closer to plain English than is FORTRAN. For example, the statement noted above might be stated in COBOL:

```
MULTIPLY RATE BY HOURS GIVING STRAIGHT-
TIME ROUNDED. DIVIDE RATE BY 2 GIVING
OVERTIME-PREMIUM. SUBTRACT 40 FROM
HOURS GIVING OVERTIME-HOURS. MULTI-
PLY OVERTIME-HOURS BY OVERTIME-PRE-
MIUM GIVING OVERTIME ROUNDED. ADD
STRAIGHT-TIME AND OVERTIME GIVING
GROSS-PAY.
```

It is obvious from the foregoing COBOL statements that such a set of instructions would be just as readable to a payroll clerk unskilled in computer fundamentals as it would be to a highly trained programer. There are certain difficulties presented by COBOL in its present state of development. Because it is relatively new, there are few standardized programs

Type of Problem	Number of Cards in Source Program		Compilation Times (in minutes)		% Increase in Compilation of COBOL Over
	FORTRAN	COBOL	FORTRAN	COBOL	FORTRAN
Job Order Cost Calculation	18	72	0.5007	0.7989	59.6
Sort Routine	25	56	0.4890	0.8108	65.8
Hourly Payroll Computation	35	115	0.5099	0.8390	64.5
Order Processing Cycle	22	288*	0.5040	1.1111*	120.46*

*Includes additional output report on rejected orders.

EXHIBIT I

Since COBOL is further removed than FORTRAN from the precise statements required by the conventions of machine-type languages, it is natural to expect longer compilation time for COBOL programs than for those written in FORTRAN.

which have been worked out and are available to COBOL users. Each installation must develop its own programs. Because of the rudimentary nature of COBOL each manufacturer attempts to add special subprograms to provide additional service for his customers. Consequently, one of the announced objectives of COBOL, that of allowing the same programs to be run on computers produced by different manufacturers, is abrogated and its flexibility correspondingly reduced.

Finally, COBOL in its present state has been designed as a language to be used in connection with batch-controlled systems.[5] This further limits the flexibility of COBOL.

Efficiency test

The difficulties considered in connection with FORTRAN and COBOL so far stem largely from an investigation of the structure of the language and from comparison of the syntactical rules. In order to gain some additional understanding of FORTRAN and COBOL, four relatively simple programs were written in FORTRAN and COBOL in order to determine the relative merits of the two languages. These programs were compiled on the IBM 7090 (operating under the IBSYS[6] system, which provides for comparable compilation of both FORTRAN and COBOL programs). Exhibit 1 above sets forth the results of the test.

The substantial difference in the length of the programs is readily apparent from this summary. Since COBOL is further removed than FORTRAN from the precise statements required by the conventions of machine-type languages, it is natural to expect longer compilation times for COBOL programs. As indicated in Exhibit 1, the COBOL programs take approximately 60 to 65 per cent longer to compile on the IBM 7090 computer. The results of the test lend strong support to the argument that FORTRAN is a more efficient language than COBOL, both in the amount of preparation required by the programer and in the amount of computer time necessary to compile identical programs.

Additional conclusions reached as a result of preparing and running these programs are summarized in Exhibit 2 on page 47.

Trend to COBOL

Another current controversy is between the use of an intermediate symbolic-type language that is machine-oriented and a problem-oriented language such as COBOL. Although it is not possible to generalize from a few specific cases, the arguments are fairly clear. The advocates of the machine-oriented languages maintain that the maximum efficiency can be obtained through using essentially one-to-one correspondence between the

programer's coding and the absolute machine-language that provides the instructions to the computer. The advocates of the problem-oriented languages maintain that this advantage of machine-oriented languages is more than offset by the reduction in programer training time and program writing and debugging. Several of the large companies are either programing in COBOL at the present time or are in the process of switching from machine-oriented to problem-oriented languages. The trend is definitely toward problem-oriented languages and, in the case of business firms, COBOL seems to be the choice.[7]

A COBOL drawback

One of the major complaints about the COBOL language has been its relatively elementary structure, requiring extensive programing to accomplish relatively simple tasks such as sorting, report generation, and preparing internal subprograms for computing logarithms, absolute values, square roots, and similar repetitive-type mathematical algorithms. A partial answer to this problem has been provided by IBM with the COBOL compiler for the IBM System/360. Five major innovations are included in the COBOL manual for this system:[8]

A report writer facility that allows considerable flexibility in designing reports

A sort option that allows internal sorting of an intermediate file through a single command

A source program library facility that provides a badly needed means of obtaining recurring programs from an alternate input unit

A means of adapting COBOL, a basic batch-controlled system language, to a system employing some type of mass storage facility

Supplementary options necessary

One of the major complaints about the COBOL language has been its relatively elementary structure, requiring extensive programing to accomplish relatively simple tasks.

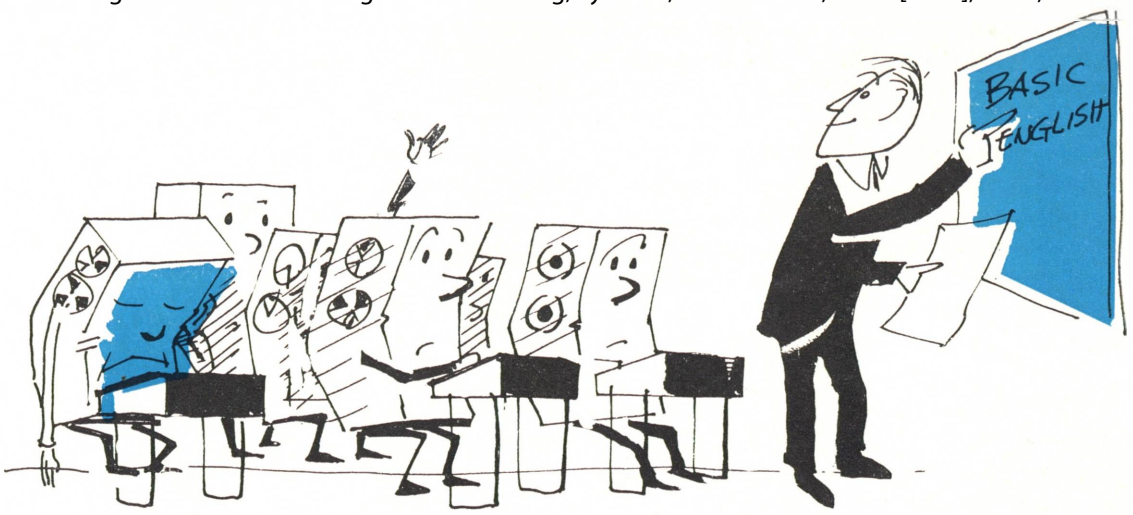
EXHIBIT 2

RELATIVE ADVANTAGES OF TWO PROBLEM-ORIENTED LANGUAGES

FORTRAN	COBOL
1. Availability of mathematical subroutines (internal functions) simplifies program preparation.	1. Programs are easier to prepare, to keypunch, and to read.
2. Availability of the SHARE* library increases the flexibilty of programing.**	2. Output editing features are better.
3. FORTRAN is more readily adaptable to statistical and operations research-type business problems.	3. Use of longer data-names (maximum of 30 characters) allows more detailed descriptions of the data.
4. Internally generated variables need not be defined in a format statement.	4. The logical structure of files, records, and elements approximates business practice.
5. Very few words are reserved in FORTRAN, allowing the programer greater freedom in the choice of variable names.	5. COBOL is more readily adaptable to batch-controlled accounting-type applications.

*This is an organization of IBM users of large-scale computers in the scientific and engineering fields. They hold regular meetings, the purpose of which is to share programs and information.

**This may be only a temporary advantage pending the development of standardized subprograms in COBOL.



In the time since the introduction of macro instructions, computers have progressed from accepting crude, detailed instructions in a form of "pidgin" English into an immensely more sophisticated and versatile understanding of a form of Basic English.

for employing telecommunication equipment as an integral part of the computer system.

Conclusions

Even though the swing seems to be in the direction of COBOL for business, it seems useful at least to suggest possible alternative programing languages that might be useful. The only other programing language in general use that seems to have any chance of succeeding is FORTRAN. It is not the purpose of this paper to make a judgment as to which language should, or will, prevail or whether PL/1 will eventually supersede both FORTRAN and COBOL. As a matter of fact, it is possible that all three languages will maintain an established place in the programing repertory of forward-looking businesses. However, certain conclusions can be drawn from this discussion:

There is a definite trend from machine-oriented languages to problem-oriented languages.

It is almost essential, given the present state of technology and programing languages, that any computers purchased should have sufficient flexibility to accept FORTRAN as well as COBOL programs with equal facility.

Strong pressures should be exerted to maintain uniformity in further developments in COBOL so that the language maintains the flexibility necessary for it to be translated by compiler programs written by each and every computer manufacturer.

---

[1] The organizations participating in the original development were the following:

Air Materiel Command, United States Air Force

Bureau of Standards, Department of Commerce

David Taylor Model Basin, Bureau of Ships, United States Navy

Electronic Data Processing Division, Honeywell, Inc.

Burroughs Corporation

International Business Machines Corporation

Radio Corporation of America

Sylvania Electric Products, Inc.

Univac Division of Sperry-Rand Corporation

[2] See, for example, R. Clay Sprowls, Editor, "Electronic Computers in Education for Business," published by Western Data Processing Center, University of California, Los Angeles, 1963, and Wayne S. Boutell, *Auditing with the Computer,* University of California Press, 1965.

[3] An algorithm may be defined as a series of well defined arithmetic steps that are followed in order to obtain a desired result.

[4] These languages are commonly referred to as machine-oriented languages although mnemonic operation codes and relative addressing distinguish these languages from absolute machine-language coding. For example, the SPS (Symbolic Programing System) and Auto-coder languages are intermediate-type languages used in connection with the IBM 1401 computer.

[5] A batch-controlled system may be defined as a system in which it is necessary to accumulate related data input (usually in sequential order) and to process this data at one time, usually updating a master file at the same time. A batch processing operation which is most familiar is the processing of a payroll.

[6] IBSYS is an abbreviation for IBM system and is a supervisory system which does the following:

a. In response to user requests (in the form of control cards), it brings a specific compiler into memory.

b. In response to user and program requests, it controls input-output operations.

[7] For example, Westinghouse, Kaiser Aluminum, and Republic Aviation

[8] IBM Systems Reference Library, *IBM System/360 Operating System, COBOL Language,* File No. S360-24, Form C28-6516-0.

https://egrove.olemiss.edu/mgmtservices/vol3/iss3/7