

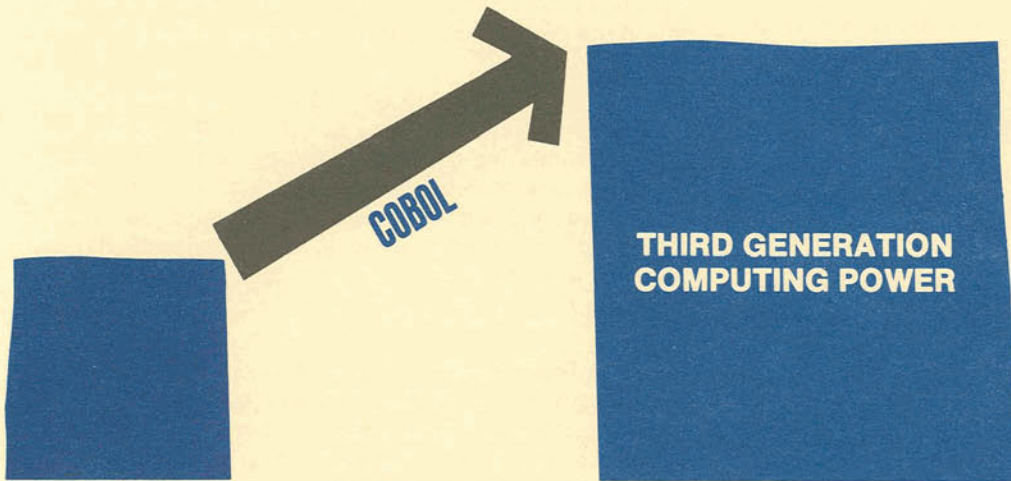


COBOL

**orientation
for
management**

TABLE OF CONTENTS

COBOL: A Powerful Capability for Third-Generation Computers	Page 3
COBOL and Six Major Decision Areas	Page 4
Some Criteria for Evaluating a COBOL Compiler	Page 8
A Complete COBOL System	Page 10
Organizing for Efficiency — With COBOL	Page 11
Honeywell Series 200 COBOL — A Brief Summary	Page 12
Anatomy of a COBOL Program	Page 13
Background of COBOL Development	Page 14



COBOL: A POWERFUL CAPABILITY FOR THIRD-GENERATION COMPUTERS

In evaluating third-generation computers, data processing management must come to grips with two issues of critical importance:

1. Just how attainable are the cost/performance advantages of a new computer? Might the cost of converting from present equipment all but offset any advantages offered by the new system?
2. Once a commitment is made for a third-generation computer, how can maximum value be obtained from every dollar spent on its operation? The familiar problems that cause inefficient computer utilization — inadequate program documentation, lack of standards, programmer turnover, inflexible programming, and the like — are not automatically solved by acquiring a new-generation computer. In fact, they may loom larger.

These two issues are not really new; similar problems existed in the past, and many techniques and tools have evolved over the years in an attempt to resolve them. Most of these tools have had only limited success. But there has been one prominent exception — COBOL, the collaborative effort by the computer user and the manufacturer to develop a standard programming medium for business data processing. COBOL has evolved over the past few years as a precise, well defined, fully mature programming capability. And in the pages that follow, you will learn how this capability can pay off in simplified transition to a new computer and continued efficient utilization of that computer.

The reader unfamiliar with COBOL will find it advantageous to turn directly to pages 13 through 15 where he will find an elementary description of a COBOL program as well as a brief COBOL history.

COBOL AND SIX MAJOR DECISION AREAS

In evaluating programming systems, data processing management is faced with six major decision areas. The costs associated with these decisions can often exceed the cost of the computer hardware itself. The following will show that COBOL is a management tool offering significant savings in all six areas.

1

INTER-SYSTEM COMPATIBILITY

Changing from one make of computer to another presents a major problem — program incompatibility. For those firms changing from 1400 series equipment, Honeywell's Liberator technique overcomes this problem by providing direct, one-time conversion of existing programs into fully compatible Series 200 programs. In other cases, however, the user's response to this problem has been a complete reprogramming job involving high costs and the loss of much time. Consequently, progressive data processing management has long asked, "What steps can be taken to make it easier to change equipment?" Many firms, large and small, have found their answer in COBOL (COMmon Business Oriented Language).

Programs written in COBOL for one computer can be run on another. In case after case, COBOL has demonstrated that it can reduce reprogramming costs substantially, even if the two computers involved have totally incompatible machine languages. Granted, COBOL programs written for one computer generally require modification before they can be compiled and executed on some other machine. However, user experience indicates that the time required to perform these modifications is measured in hours, not days or weeks.

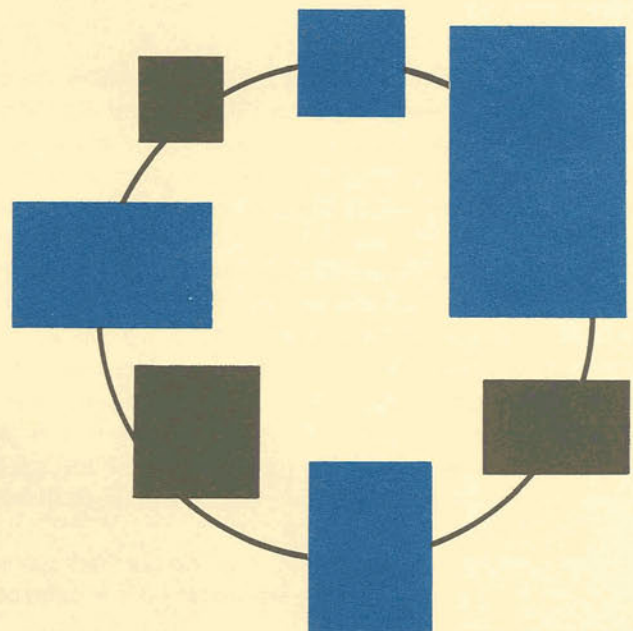
For example, in a recent changeover to a Honeywell 200 system, 350 of the user's existing COBOL programs were modified by one man in just nine weeks — an average of roughly one hour per program. For virtually every program, modifications were minor. There was no need to rethink or rework the basic logic of the programs. The payoff for the user came just hours after receiving the 200. The first production run was completed only 11 hours after the machine had been turned over to them. On the very next day, they processed the entire plant payroll, on time and with no help from their old computer. Significantly, and typically, the user found that Series 200 COBOL not only facilitated transition to the new equipment but automatically improved the efficiency of the old programs as well. In general, the Series 200 compiler-produced programs requiring about 4,000 fewer characters of core storage than programs created by the user's earlier COBOL compiler.

To understand why modifications to COBOL programs are essentially a minor effort usually involving only a few hours work, you must first be familiar with the structure of a

COBOL program. A COBOL program is written in four divisions, of which the most important are the Data Division, which describes the data the program is to process, and the Procedure Division, which states the logical steps that the program is to follow in processing the data. These two divisions are completely independent of each other; thus changes can be made to the Data Division without affecting the program logic described in the Procedure Division. And if changes are made, chances are that they will be made only in the Data Division for the following reasons:

Both divisions are said to be machine-independent; the descriptive information in each is not directly related to the logic or processing capabilities of any particular computer. Therefore, it is possible to approach 100% compatibility in the Procedure Division since the procedural steps of the program do not reflect the way in which any particular computer will execute them. If the Procedure Division is valid for computer A, it will be equally valid for computer B.

The Data Division presents a slightly different situation. In describing the data to be processed by a particular computer, it is desirable for purposes of maximum efficiency to lay out file structures in a manner best suited to that computer. Similarly, when changing to another computer, it is almost always desirable to restructure the data files so as to take advantage of such key factors as word size, sign conventions, and data codes. This is not to say that most COBOL compilers cannot handle a Data Division which was described for a different machine. However, since only the Data Division need be modified and since most modifications to the Data Division are minor, the small amount of time spent in making these changes is a sound investment in greater efficiency.

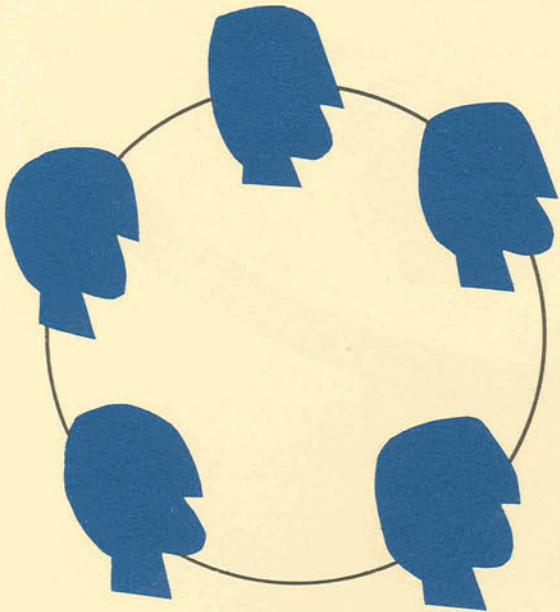


2

STANDARDIZATION

Being a standard language, COBOL has important implications for the availability and assignment of programmers. The user is often faced with these questions: What effect will turnover have on the installation? Where will new programmers come from? If the user loses a programmer does it mean that he must find another programmer with the same specialized experience? What are the chances of finding a programmer with experience on a particular machine? Can a new programmer take over in the middle of a job, or will he have to start from scratch because he cannot interpret the work already accomplished?

COBOL's standardization offers an effective solution to all of these problems. Regardless of the machines they have programmed, all COBOL programmers use the same language elements and adhere to the same procedural rules. Thus any programmer with experience in COBOL can pick up the reins right in the middle of the job. The advantages offered by COBOL's standardization can be further appreciated by what a current user of Series 200 COBOL had to say in this regard: "Rapid expansion of our programming staff was facilitated by the ability to hire programmers with any type of COBOL experience and to have them engaged in productive programming with less than eight hours of orientation to the new system."



3

INTRAPROJECT COMMUNICATION

Good intraproject communication is essential to sound and widespread understanding of an installation's operations. COBOL is a well defined set of simple English-language elements with documentation to match and thus assures a quick grasp of all aspects of a project even by non-technical personnel. A user of Series 200 COBOL found that it played a major role in improved communication: "Reduced conversion time was realized as a result of improved communication between programmers working on related programs."

COBOL language elements constitute only part of a COBOL program. Many names and terms must be used to describe the various data files. COBOL offers a precise communication medium that can introduce corporate standards into data nomenclature. Where several programmers are involved in preparing programs for a particular job, standardized data file nomenclature will enhance intraproject communication.



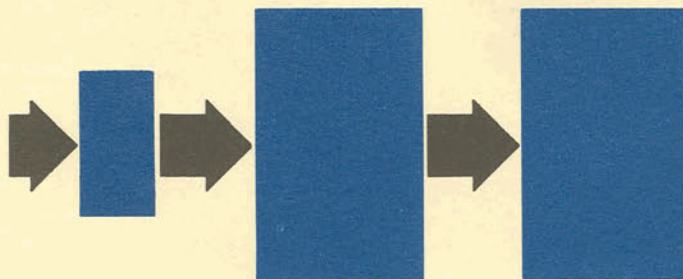
4

PROGRAM TESTING

COBOL substantially reduces the cost and time associated with checking for and correcting errors because the COBOL compiler always produces programs which are mechanically correct. During compilation, the compiler automatically detects and pinpoints clerical errors such as keypunching mistakes, transposed digits, and improper addressing information. Since the compiler catches all clerical errors, the programmer need only check for logical errors; for example, he may have coded an ADD CORRESPONDING statement when he meant to code MOVE CORRESPONDING.

Experience indicates that a programmer can completely test a COBOL program with a total of three compilations. As one user put it, "When testing COBOL programs, our people find little need to use that shop-worn expression, 'almost ready, just one more debugging run.'" Another user found that after switching to COBOL, "debugging time was reduced 50%."

The first compilation is used to scan for clerical errors. (Honeywell COBOL compilers accomplish this quickly by using a fast diagnostic-scan feature which performs an analysis and produces a listing of the program but does not waste time on a full compilation.) The second and third compilations are devoted to tracking down any logical errors in the program.



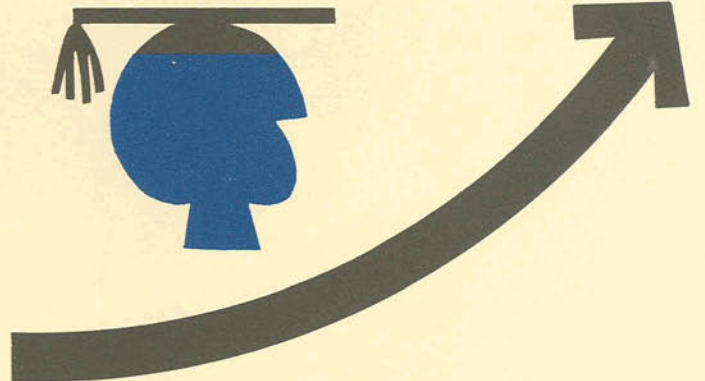
5

PROGRAMMER TRAINING

Programmer training (or retraining in the case of a change-over to a new computer) can be expensive and time-consuming even when experienced programmers are involved. Even after formal training is completed, it may be some time before the programming staff can attain a satisfactory level of productivity, so this time should be figured into the cost of programmer training.

The company with COBOL-trained programmers faces no such problems. COBOL is machine independent: if a programmer has learned COBOL, he can apply it to any machine having a COBOL compiler.

COBOL is the ideal language for new personnel with no programming experience whatsoever. Its skills can be learned in modular steps. The beginner does not have to learn the entire language before he becomes productive. He can use only a small part of the language and still create useful programs. As he develops his skills he can take on more of the language and thereby improve the efficiency of his programs. Many companies have found that this modularity has enabled COBOL programmers to be productive many weeks in advance of programmers being trained in a conventional assembly language.



6

DOCUMENTATION

Clear documentation, inherent in COBOL, is vital to intra-project communication and indispensable where one programmer is taking over a job from another. It provides a fuller picture of the program's logic since the English-language procedural sentences written by the programmer clearly describe his intent. With COBOL there is no need to prepare documentation after coding the procedure: the coding is its own documentation.

What's more, a COBOL compiler can produce valuable diagnostic information in the form of program listings. Honeywell Series 200 compilers produce an array of carefully designed listings containing such information as diagnostic messages about the source program, syntax analysis, and cross references between the COBOL language statements and their machine-language counterparts. Here is what one user had to say about the value of this documentation: "The elapsed time from the first compilation to the time that a program is put into production has been significantly reduced by the excellent diagnostic statements provided by the Honeywell COBOL D compiler."



SOME CRITERIA FOR EVALUATING A COBOL COMPILER

1 LANGUAGE ELEMENTS

How many features of the approved COBOL language have been included in the compiler? No manufacturer has attempted to implement the entire language. Instead, subsets of the language have been produced which strike a balance between language power and realistic equipment configurations for using these subsets. As a result the user must carefully evaluate what might be termed the "richness" of a particular subset of the language. Are enough language elements provided to do the job and does the compiler operate in a realistic equipment configuration? For example, Honeywell's COBOL D offers 270 language elements with a compiler requiring 16,000 characters of core memory.



2 LANGUAGE INTEGRITY

Has the manufacturer implemented both the letter and the intent of the approved COBOL language? Has he preserved the integrity of the language elements or have modifications been made to suit particular machine requirements, modifications which will render the COBOL programs inefficient on other machines?

3 COMPILER SPEED

How fast is the compiler? Does it offer an optimum balance between fast compilation and efficient machine-code output, or does it sacrifice one for the other? A compiler offering this balance can be one of the most valuable tools in your installation. With a fast compiler, such as Honeywell's COBOL D which can compile an average program in less than two minutes, all program changes can be made at the English-language source-program level. Recompilation is justifiable even for the most minor changes. The combination of source-program maintenance and fast compilation appreciably reduces the time required for program changes and provides a strong incentive to avoid the costly practice of patching machine-language programs.

4 OBJECT CODE EFFICIENCY

How efficient is the machine code produced by the compiler? How does this code compare with that produced by an average programmer using assembly language? Is much more coding produced by the compiler than is necessary? Many of the newer compilers can produce code which, for all practical purposes, is equal to assembly language in terms of running efficiency. (In terms of measurable efficiency, the average Honeywell COBOL program achieves 85-90% of the efficiency of an assembly-language program.) Though it is true that the efficiency of a COBOL program can be exceeded by that of a program written by a highly skilled assembly-language programmer, the question arises as to how many highly skilled programmers are available or even needed at the average installation.

5 MANUFACTURER SUPPORT

What support is offered by the manufacturer? In addition to the usual programmer reference texts, does he provide training aids such as programmed instruction texts, or special aids such as guidelines for optimizing object-code efficiency? Honeywell provides all of these. In the case of object-code guidelines, for example, Honeywell provides ample documentation showing how many machine instructions are generated for each variation of every COBOL verb. With such information, programmers can readily determine the efficiency of the various alternatives available to them. Extensive manufacturer support such as this can mean big savings.

4

10011001010

5



A COMPLETE COBOL SYSTEM

Many computer users tend to associate the term COBOL with just two elements: a manufacturer's selection of COBOL language elements, and the compiler provided for translating a COBOL program into its machine-language equivalent. Actually these two elements are only a part of what might be viewed as a total COBOL system. In effect, the language and the compiler can be all but useless if an extensive array of features is not provided to assist in the preparation, maintenance, and usage of COBOL programs.

The depth of support offered by the manufacturer can not be measured by evaluating just the language and the compiler. To do so would be like attempting to determine the sturdiness of a tree by looking only at the leaves and branches. The leaves and branches may present a pretty image but the tree may be in danger of toppling over because of a shallow root structure. As such, the structure of a COBOL system is analogous to the structure of a tree. The trunk and branches of the tree represent the COBOL language and compiler, while the root structure represents the supporting preparation, maintenance, and execution aids. And it isn't until one digs into the supporting root structure that the actual amount of support can be determined.

Following is a list of systems features which are all-important in the efficient support of COBOL but which seldom receive the consideration they deserve. These features are found in all Honeywell COBOL systems above the basic (8K) design level, which includes a subset adequate for a small equipment configuration.

PROGRAM PREPARATION

- Source-language file maintenance
- Library copy facilities
- Batch compilation
- Load-and-go facility
- Fast diagnostic scan
- Composite program listings with imbedded diagnostics
- Operating system control
- Environmental adaptability

PROGRAM EXECUTION AND MAINTENANCE

- Object-program file maintenance
- Job-oriented testing
- Operating system control
- Debugging facilities
- Object-time file relocation
- Dynamic control of input/output facilities



ORGANIZING FOR EFFICIENCY — WITH COBOL

COBOL is a unique capability and thus offers unique opportunities for organizing your installation. Following are a few of its more important advantages in this respect.

MORE PRODUCTIVE USE OF PROGRAMMERS

Much of programming work is strictly clerical. While with other programming systems there is no way of separating the clerical aspects of programming from the technical, COBOL can free the programmer from much of this mechanical work to spend his time more productively.

Usually the programmer draws a flow chart of the procedure the program is to follow, and from this flow chart he writes the program on a coding form. By using COBOL sentences in the flow chart, the programmer can reduce coding to little more than transcribing statements from flow chart boxes onto the coding form, a mechanical task easily handled by non-technical personnel.

EFFICIENT PROGRAM PREPARATION

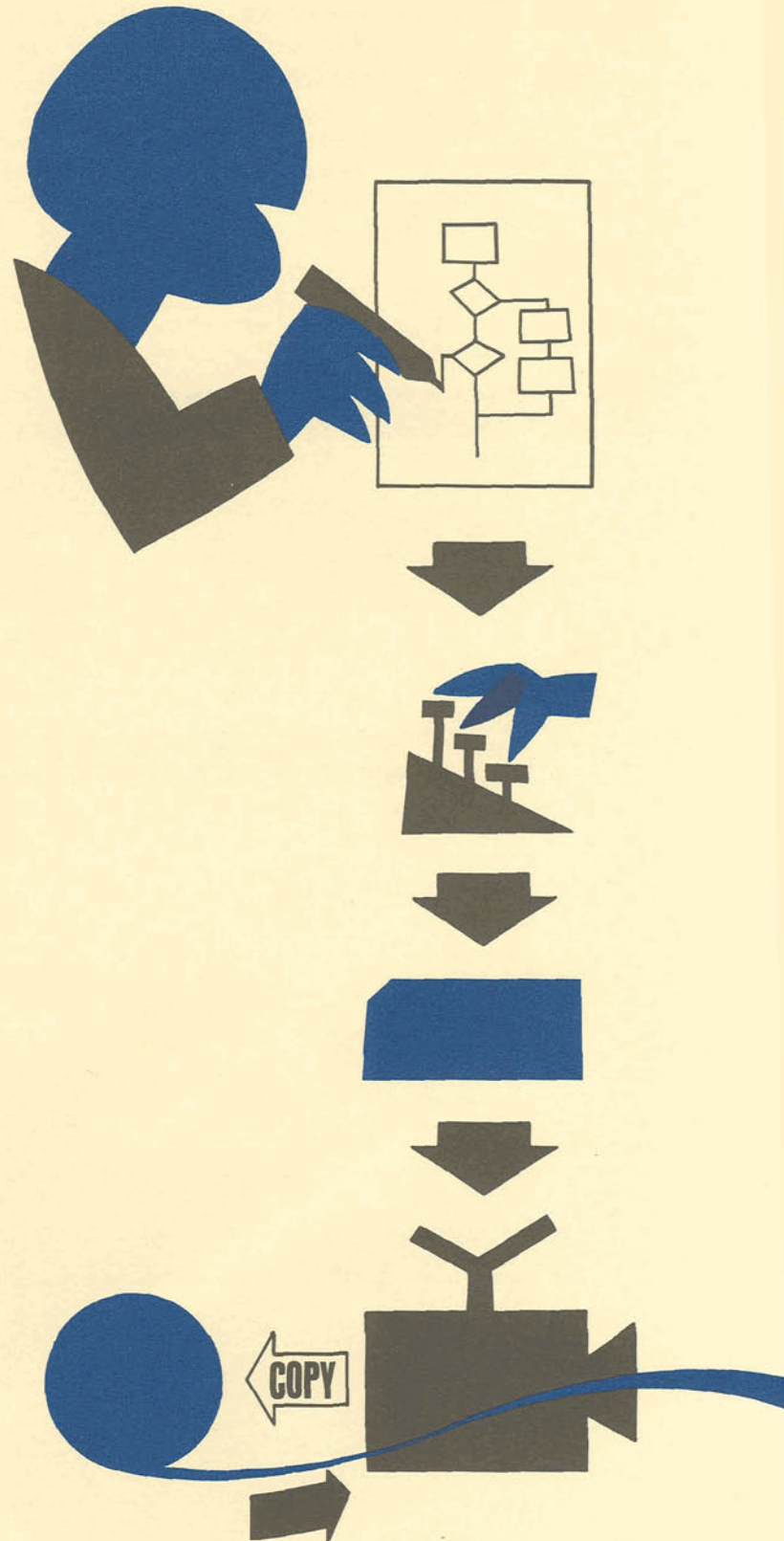
DIRECT KEYPUNCHING — Taking COBOL's simplicity a step further, formal coding can be eliminated altogether. Some COBOL users are finding that keypunchers can punch COBOL source programs directly from flow charts. An effective safeguard against keypunching errors can be found in the fast preliminary diagnostic scan offered by most new high-performance compilers.

DATA DIVISION LIBRARY — Every COBOL program has a Data Division which contains the structure and nomenclature of the data files to be used by that particular program. By creating a library of often used Data Divisions, the user can avoid the necessity of rewriting the Data Division every time he is going to use the same file for a different program. By means of the COPY verb, the appropriate Data Division descriptions can be automatically inserted into the programs during compilation. The library can be updated as required, with new listings distributed to programmers for the purpose of writing Procedure Divisions.

STANDARD TERMINOLOGY IMPROVES COMMUNICATION

The COBOL language provides only the verbs and connectives used in a program. The majority of terms found in a COBOL program are Data Division names describing files, records, and fields. These terms are often devised by the various programmers without coordination and with the inevitable result that several programmers refer to the same data using different names.

By placing the Data Division library under control of one person, the systems analyst for example, the installation can enforce standardization of terminology and thus improve communication between all programmers working on a project. Any changes to the Data Division library can be cleared through the controlling authority, and all personnel involved can be informed of the changes by way of new Data Division listings.



HONEYWELL SERIES 200 COBOL — A BRIEF SUMMARY

Now that you have seen what COBOL can do, how a powerful implementation of COBOL can be assured, and how a COBOL system is structured, take a look at how COBOL can be put to work.

Honeywell offers five levels of COBOL — a range of capabilities to fit virtually every data processing need. The following table summarizes basic characteristics important to your evaluation of COBOL.

COMPILER	B†	D	H	J	K
Core Memory Required (Characters)	8K	16K	32K	65K	131K
Language Elements	133	270	346	346	346*
Minimum Peripheral Requirements ‡	2 Tape Units Card Reader Card Punch Printer	4 Tape Units Card Reader Card Punch Printer Advanced Programming & Edit	4 Tape Units Card Reader Card Punch Printer Advanced Programming & Edit	6 Tape Units Card Reader Card Punch Printer Advanced Programming & Edit	6 Tape Units Card Reader Card Punch Printer

Program Preparation Features

Source-language file maintenance?	No	Yes	Yes	Yes	Yes
Library copy facilities?	No	Yes	Yes	No	Yes
Batch compilation?	No	Yes	Yes	Yes	Yes
Load-and-go?	Yes	Yes	Yes	Yes	Yes
Fast diagnostic scan?	Yes	Yes	Yes	Yes	Yes
Composite program listing with imbedded diagnostics?	Yes	Yes	Yes	Yes	Yes
Monitor control?	Yes	Yes	Yes	Yes	Yes
Environmental adaptability?	No	Yes	Yes	Yes	Yes

Program Execution & Maintenance Features

Object program file maintenance?	No	Yes	Yes	Yes	Yes
Job-oriented testing?	No	Yes	Yes	Yes	Yes
Monitor control?	Yes	Yes	Yes	Yes	Yes
Debugging facilities?	No	Yes	Yes	Yes	Yes
Object-time file relocation?	No	Yes	Yes	Yes	Yes
Dynamic control of I/O channels?	No	Yes	Yes	n/a	n/a

† The COBOL Compiler B was designed to run on small-scale, 8K systems. Thus, while it is the first effective COBOL compiler to be offered for small systems, its operating features are necessarily restricted.

* Additional elements under consideration although not presently specified.

‡ Mass Memory Files can be used in lieu of tape units.

ANATOMY OF A COBOL PROGRAM

The following is a brief description of the structure of a COBOL program. It illustrates the ease of using this business-oriented language, as well as the functional independence of the various parts of the program. For the non-technical reader, it offers further insight into the significance of the major concepts presented on the preceding pages.

A manufacturer's implementation of COBOL consists of two parts: a language and a compiler. The language is used by the programmer in writing his *source* program. The compiler is a program, supplied by the manufacturer, that processes the COBOL source program to produce (i.e., compile) a machine-language *object* program.

A COBOL source program has four divisions:

The Identification Division contains the name of the program and other identifying information as desired.

The Environment Division specifies the computer on which compilation is to be accomplished and the computer configuration on which the machine-language object program is to run.

The Data Division describes the data files with which the object program is to work.

The Procedure Division states the logical steps that the object program is to follow in processing the data.

Following are more detailed descriptions of each division.

IDENTIFICATION DIVISION

The entries in the Identification Division are simple and straightforward. This division is the first part of the documentation that is printed during compilation of the source program. The foremost entry in this division is the name of the program. Other information, such as the date the program was written, the name of the programmer, and a brief description of the program function can also be included.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. INVENTORY-PROGRAM.  
DATE-WRITTEN. JANUARY 10, 1966.  
AUTHOR. SAM JONES.  
SECURITY. CONFIDENTIAL TO COMPANY.
```

DATA DIVISION

The Data Division includes descriptions of the data files that the object program is to manipulate or create and the individual logical records which make up these files. Since the characteristics of the data are described in relation to a standard problem-oriented format rather than an equipment-oriented format, this division is to a large extent computer-independent. Thus, while 100% compatibility among computers cannot be absolutely assured, careful planning in the data layout will permit the same data descriptions, with minor modifications, to apply to more than one computer.

```
DATA DIVISION.  
FILE SECTION.  
FD  
LABEL RECORDS ARE STANDARD;  
VALUE OF IDENTIFICATION IS "TRANS";
```

ENVIRONMENT DIVISION

This part of the source program specifies the equipment being used. It names the computer on which the source program is to be compiled and describes the computer on which the object program is to run. Memory size, number of tape units, printers, and so on are among the entries that can be made when describing the object-program computer. Problem-oriented names can be assigned to particular pieces of equipment. Because the Environment Division deals entirely with the specifications of the equipment being used, it is largely computer-dependent.

```
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER. HONEYWELL-200.  
OBJECT-COMPUTER. HONEYWELL-200; 4 TAPE-UNITS;  
SEGMENT-LIMIT IS 10.  
SPECIAL NAMES. CONSOLE TYPEWRITER IS ERROR PRINTER.
```

PROCEDURE DIVISION

The Procedure Division specifies the steps that the programmer intends the object program to follow in processing data. These steps are expressed in terms of meaningful English-language words, statements, sentences, and paragraphs. This division is essentially computer-independent; that is, the information appearing in this division is independent of the machine logic of any particular computer.

The outline of the procedures required to process the data is normally diagrammed in a flow chart. By using COBOL sentences in the flow chart, the programmer can simply transcribe them to produce his Procedure Division. Some firms have found that such transcription is unnecessary because keypunching can be performed directly from the flow charts (see preceding section, ORGANIZING FOR EFFICIENCY — WITH COBOL). This procedure saves time and money and improves program documentation.

```
PROCEDURE DIVISION.  
BEGIN SECTION 15.  
OPEN-FILES. OPEN INPUT PURCHASE-FILE; OUTPUT  
TOTAL FILE  
PROCESSING SECTION 1.  
READ-LOOP. READ PURCHASE-FILE; AT END GO TO  
END-PROCESSING. IF STOCK-NUMBER IN  
PURCHASE-RECORD IS NOT NUMERIC GO TO BAD-STOCK-NUMBER.
```

BACKGROUND OF COBOL DEVELOPMENT

In 1959, representatives of the various computer manufacturers met with the world's largest computer user — the Department of Defense — to prepare the groundwork for development of a common programming language. The need for such a language had long been evident. Dissimilarities between the various computers, and between the languages used to write programs for these computers, were a source of enormous costs to users. In fact, the advantages to be gained by moving to a more powerful computer of another manufacture were often completely offset by the costs of reprogramming and retraining.

The inevitable outgrowth of this situation was the demand for a common language. A committee of computer manufacturers and users was formed to cooperate on the development of a standard set of programming terms, the COBOL language, and the manufacturers agreed to develop individual compilers which would translate programs written in this language into programs executable on their own machines.

THE COBOL LANGUAGE

The first version of the COBOL language was published in 1960. As might be expected in such a large undertaking, this first version was by no means perfect. The language was ambiguous and covered only limited areas within the overall data processing environment. The fact that some of the compilers created by the various manufacturers interpreted phrases and sentences differently made it apparent that certain language clarifications would have to be made before COBOL could be satisfactorily applied.

In the few short years since publication of the first version of COBOL, development of the language has made tremendous progress to the point where it is today a fully developed and powerful capability. Over 200 clarifications have been made to the language and several major capabilities have been added, including sort verbs, the report writer, table handling verbs and a mass storage capability.

COBOL is now widely accepted by the computer-using community. Significantly, official recognition has come from the American Standards Association, which has accepted COBOL as a candidate for an American standard programming language, and from the European Computer Manufacturer's Association which is currently considering COBOL for adoption as an international programming standard.

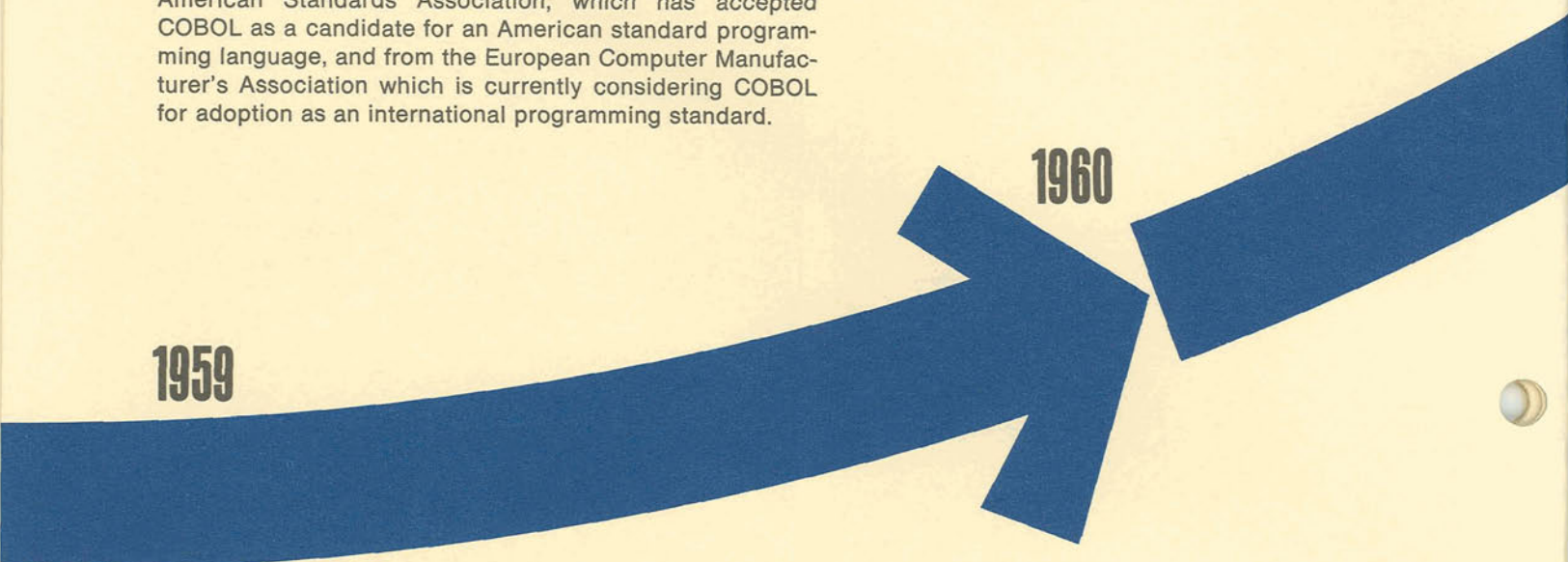
THE COBOL COMPILERS

Compiler technology was not sufficiently advanced in 1960 to cope with a language such as COBOL. Those few compilers which attempted to translate a full COBOL language set proved to be unwieldy, slow, and inefficient. Compile times were often measured in terms of hours — typically, one to two hours were required to compile a single program — and program testing was a serious drain on computer time. Another defect of early compilers was the inefficiency of the machine code they generated. It was commonplace for a compiler to generate five times as much code as would have been generated using assembly language. Still another drawback was the inconsistency between the COBOL compiler and the other software associated with a particular system.

But COBOL compilers have also taken big strides since 1960. Comprehensive sets of language are today being implemented by extremely fast compilers. Compile times are now measured in minutes, even seconds. Program reliability has increased several times since faster compile times permit more checkout runs in a given period of time. Machine-language efficiency is also vastly improved: programs produced by today's Honeywell COBOL compilers approach 90% efficiency (i.e., they are 90% as efficient as they would be if programmed using a conventional assembly language). In addition, COBOL compilers and the programs they produce are now integral parts of extensive operating systems.

1959

1960



**COBOL TODAY —
A MATURE PROGRAMMING CAPABILITY**

COBOL is as dynamic as computer technology itself, and tomorrow's COBOL will be even more powerful than today's. For example, studies are being carried on to investigate such things as extensions of the COBOL language, improved source-language debugging tools, and better techniques for program segmentation. However, today's COBOL is without a doubt a mature and powerful programming system. It has arrived at this point by way of the only route possible for such an ambitious undertaking: years of collective and individual effort on the parts of several major computer manufacturers and users.

Today, data processing management is finding that evaluation of COBOL is shifting from, "Can we afford to use it?" to, "Can we afford not to use it?"

TODAY



102646124

Honeywell
ELECTRONIC DATA PROCESSING

SALES OFFICES AND DATA CENTERS
IN PRINCIPAL CITIES OF THE WORLD