COBOL

By R. F. Clippinger

The description of COBOL as of late 1960 is well handled by an article by Jean Sammet in the *Annual Review in Automatic Programming*, Volume 2. This discussion will, therefore, stress developments since that time. The period from March 1960 to May 1961 was used by the COBOL Committee to clarify COBOL 60 and remove ambiguities as far as possible. This resulted in the publication of the second COBOL report entitled: "COBOL Report to Conference on Data Systems Languages, including Revised Specifications for a Common Business Oriented Language (COBOL) for Programming Electronic Digital Computers, Department of Defense 1961," and is available from the Superintendent of Documents, US Government Printing Office, Washington 25, DC, price $1.25.

By December 1961 there were twelve manufacturers in the United States and three foreign manufacturers committed to prepare compilers for thirty-five makes and models of computers. These compilers vary in regard to the faithfulness with which they have attempted to implement the required features, and in the extent to which they have proceeded to implement certain elective features. A few of these compilers are in systems test (RCA, Sperry Rand, Air Force Logistics Command, National Cash Register, IBM 1410), others are in various stages of development. The total volume of completely checked-out applications running today in COBOL language is negligible, representing less than 1% of the great volume of data processing going on in the United States today. A fairly significant number of computer users are committed to the use of COBOL, many of them in blind acceptance of all the advantages of compilers without any understanding of the limitations upon these advantages. There has already been a certain amount of vivid disappointment on the part of some of the earliest users, because they were expecting too much, and the compilers they were working with were not working well yet and had not been improved to the extent possible. Only heavy-volume usage of COBOL, with compilers that have been tuned up to a much higher level of efficiency, will produce the perspective which will enable us to evaluate with certainty its success. This heavy volume will not be available before the end of 1963.

Meanwhile, assuming the ultimate success of COBOL, the COBOL Committee has proceeded, in the period from May 1961 to April 1962, to work on various extensions to COBOL, three of which have now been adopted, and are in final editing, getting ready for publication in July 1962 as "COBOL 61 extended." The present paper makes no attempt to provide an authoritative description of these extensions, which will be defined by the July official publication. The only purpose here is to give general information about the nature of these changes.

Generalized arithmetic verbs

The five arithmetic verbs, ADD, SUBTRACT, MULTIPLY, DIVIDE and COMPUTE, have all been expanded to permit the specification of two or more result fields, each with its own ROUNDED option. This expansion applies to both result fields used as operands, and to result fields not used as operands (i.e. following the word GIVING).

The expanded formats of the ADD, MULTIPLY, and COMPUTE verbs with multiple result fields not used as operands, are as follows:

ADD $\left\{\begin{matrix}\text{literal-1}\\ \text{data-name-1}\end{matrix}\right\}$, $\left\{\begin{matrix}\text{literal-2}\\ \text{data-name-2}\end{matrix}\right\}$ $\left[, \left\{\begin{matrix}\text{literal-3}\\ \text{data-name-3}\end{matrix}\right\} \ldots\right]$ *GIVING*

data-name-*m* [*ROUNDED*] [, data-name-*n* [*ROUNDED*] . . .]

[; ON *SIZE ERROR* any imperative statement]

MULTIPLY $\left\{\begin{matrix}\text{literal-1}\\ \text{data-name-1}\end{matrix}\right\}$ *BY* $\left\{\begin{matrix}\text{literal-2}\\ \text{data-name-2}\end{matrix}\right\}$ *GIVING* data-name-*m*

[*ROUNDED*] [, data-name-*n* [*ROUNDED*] . . .] [; ON *SIZE ERROR* any imperative statement]

COMPUTE data-name-*m* [*ROUNDED*] [, data-name-*n* [ROUNDED] . . .] $\left\{\begin{matrix}\textit{FROM}\\ =\\ \textit{EQUALS}\end{matrix}\right\}$

$\left\{\begin{matrix}\text{data-name-}k\\ \text{literal-1}\\ \text{formula}\end{matrix}\right\}$ [; ON *SIZE ERROR* any imperative statement]

The SUBTRACT format differs from the ADD format only in that a "FROM data-name-*k*" appears in front on the word "GIVING" and the DIVIDE format differs from the MULTIPLY format only in using the word "INTO" instead of "BY." In each case the expansion lies in allowing "data-name-*n* . . ." along with "data-name-*m*" as a result field.

The expanded formats of the ADD and MULTIPLY verbs with multiple result fields used as operands are as follows:

ADD { literal-1 / data-name-1 } [, { literal-2 / data-name-2 } . . .] *TO* data-name-*m*

[*ROUNDED*] [, data-name-*n* [*ROUNDED*] . . .] [; ON *SIZE ERROR* any imperative statement]

MULTIPLY { literal-1 / data-name-1 } BY data-name-*m* [*ROUNDED*] [, data-name-*n*]

[*ROUNDED*] . . .] [; ON *SIZE ERROR* any imperative statement]

The SUBTRACT format differs from the ADD format in the above case only in using the word "FROM" instead of "TO." The DIVIDE format differs from the MULTIPLY format in the above case only in using the word "INTO" instead of "BY." Again the expansion lies in allowing "data-name-*n* . . ." along with "data-name-*m*" as the final operand and result field.

A further extension to COBOL in the arithmetic area involves the use of the CORRESPONDING concept in ADD and SUBTRACT statements. The format is as follows:

ADD *CORRESPONDING* data-name-1 *TO* data-name-2 [*ROUNDED*]

[; ON *SIZE ERROR* any imperative statement]

Data-name-2 is a result field used as an operand. In the SUBTRACT format, "TO" is replaced by "FROM." The meaning of the "CORRESPONDING" is that an individual ADD statement is performed for each pair of numeric fields, subordinate to data-name-1 and data-name-2, which have matching names. Two vectors, therefore, can be added by one ADD CORRESPONDING statement.

Sorting

Influenced to a considerable extent by FACT and by Honeywell, the COBOL Committee has extended COBOL 61 to include sorting. The changes required to incorporate sorting affect the data division and the procedure division. The form of a sort statement in a COBOL program will be:

SORT file-name ON { DESCENDING / *ASCENDING* } *KEY*
data-name-1 [, data-name-2 . . .]
[; ON { *DESCENDING* / *ASCENDING* } *KEY* . . .]

{ *INPUT PROCEDURE* IS section-name-1 / *THRU* section-name-2 / *USING* file-name-1 }

{ *OUTPUT PROCEDURE* IS section-name-3 / *THRU* section-name-4 / *GIVING* file-name-2 }

The KEY clauses permit the specification of any number of sort keys, and each key can be sorted from low to high value (ASCENDING) or from high to low value (DESCENDING).

The USING and GIVING options allow different names to be assigned to a file before and after sorting. The INPUT PROCEDURE allows any amount of modification, selection or creation of records during the initial pass of the sort operation, and the OUTPUT PROCEDURE allows similar modification and selection during the final merge pass of the sort operation. The inclusion of the sort verb in COBOL eliminates the necessity to leave the COBOL language and enter some local language, and thus constitutes a giant step further towards commonality.

Report writer

Report writing is another extension of COBOL which was heavily influenced by FACT, though FACT was not the only influence since SURGE and other programming systems have incorporated report writers. The report writer is a major extension of COBOL which will greatly facilitate for the programmer the job of creating reports. There are approximately fifty pages of new material that will be inserted in the new July 62 COBOL Specifications manual to define the report writer. The Report Section constitutes a fourth section in the Data Division. Three new verbs are provided in the Procedure Division: INITIATE, GENERATE, TERMINATE. These verbs are to report writing as OPEN, WRITE, CLOSE are to file processing. Thirty-seven new reserved words are introduced by the report writer —such words as COLUMN, CONTROLS, DETAIL, FINAL, FOOTING, GROUP, HEADING, LINE COUNTER, PAGE, PAGE COUNTER, REPORT, are suggestive. Reports are named in the file description entries or described in the Report Section. Cross-

referencing between the description of the report, description entry and the associated file description entry is handled by a clause of the form:

REPORT(S) { IS / ARE } data-name-1 [; data-name-2 . . .]

placed in the file description entry. The approach taken by the COBOL report writer is to separate the physical aspects of the report format from the conceptual characteristics of the data to be included in the report. The former are described by RD entries (Report Name Description Entry) of the form:

Option 1:

RD data-name-1 [WITH *CODE* mnemonic-name-1]
COPY library-name

Option 2:

RD data-name-1 [WITH *CODE* mnemonic-name-1]
[; *CONTROL* (S) . . .] [; *PAGE* LIMIT (S) . . .]

which name the report and determine the structure of the page, giving the number of physical lines per page and the limitations for presenting specified headings, footings, and details within a page structure. Data items which act as control features during presentation of the report are specified. The form of the CONTROL(S) clause is:

[; *CONTROL(S)* { IS / ARE } { *FINAL* / data-name-1 [, data-name-2 . . .] [, data-name-*n*] / *FINAL* , data-name-1 [, data-name-2 . . .] [, data-name-*n*] }]

and the form of the PAGE LIMITS clause is:

[; *PAGE* LIMIT(S) { IS / ARE } integer-1 *LINES*
, [*HEADING* integer-2] [, *FIRST DETAIL* integer-3]
[, *LAST DETAIL* integer-4] [, *FOOTING* integer-5]]

The report group description entries have the following format:

Option 1:

nn [data-name-1] *COPY* data-name-2
[FROM *LIBRARY*]

Option 2:

nn [data-name-1]
[; *LINE* NUMBER . . .]
[; *SIZE* . . .]
[; USAGE . . .]
[; CLASS . . .]
[; *NEXT GROUP*]
; *TYPE* . . .

Option 3:

nn [data-name-1]
[; *COLUMN* NUMBER . . .]
[; *GROUP* INDICATE]
[; CLASS . . .]
[; editing clauses]
[; *JUSTIFIED*]
[; *LINE* NUMBER . . .]
[; *POINT* LOCATION]
[; *SIGNED*]
{ ; *SOURCE* . . . / ; *SUM* . . . / ; *VALUE* . . . }
[; *PICTURE*]
[; *RESET* . . .]
[; *SIZE* . . .]
[: USAGE . . .]

A report group may be an elementary item, a line of print, or a series of lines, hence the use of a level number and a TYPE description which clarifies whether the report group is a report heading, page heading, overflow heading, control heading, detail line, control footing line, overflow footing, page footing, or report footing. The level number gives the depth of the group; the TYPE description defines the type of group. Any groups belonging to other groups have the same TYPE description. Report groups are created in object time as a result of the use of the verb GENERATE in the Procedure Division. Many of the expressions used in the report group description entry are sufficiently suggestive that one can guess their use. Space does not permit elaboration at this point. Suffice it to say that everything needed to create a report of considerable flexibility, including the generation of lines and tables, tabulated fields, headings, footings, etc., is available.

It is expected that users of COBOL 61 will be pleased that COBOL 61 extended now includes the SORT Verb and the Report Writer.

Next stage of COBOL development

The COBOL Committee has determined that there will not be a publication of the language specifications called COBOL 62. During the rest of 1962, part of the committee will turn its attention back on COBOL 61 extended, and revise, polish, remove ambiguities, and clarify questions of interpretation. This maintenance review will lead to what will be called COBOL 63, which should be basically compatible with COBOL 61 extended. Meanwhile, a few more extensions will be worked on, notably table handling and possibly provision for the use of bulk files.

Elective features

In the COBOL 61 Manual, some features of the language are designated "elective" (as opposed to "required") meaning that an implementator can post-

pone the implementation of these elements in order to get a COBOL system "on the air." The elements of required COBOL thus define the minimum language acceptable in order to consider a system as a COBOL compiler.

Some users tend to want every elective feature changed in status, to become a required feature. Others realize that the inclusion of more features in a compiler automatically make the compiler larger, which has the tendency to make it compile more slowly. It is, of course, much easier to implement the most elaborate forms of COBOL on machines of large capacity and high speed. It is probably not reasonable to implement some of the elective features on the smallest computers for which COBOL will be provided. It is anticipated, therefore, that certain electives will change status, but not necessarily all of them, and that the changes of status will come slowly. Clearly any statement of this form represents the author's opinion which may not be shared by others.

Honeywell COBOL

Honeywell is hard at work implementing COBOL for the 400. The complement of equipment which it specifies is one Card Reader, four Magnetic Tapes, 2,048 words of Core Memory, and one Printer; the Compiler will take advantage of additional tape and memory. Initially the compiler will implement all required and many elective features of COBOL, and will eventually implement the extensions discussed in this paper. Honeywell is about to begin the implementation of COBOL for the 800 and 1,800. Since it is doing COBOL for three machines, it has chosen an implementation technique rather different from that used for FACT—a technique with which we expect to shorten delivery time and reduce cost to us for the compiler development. We prefer to wait until the compilers are working before we discuss these techniques.

Information algebra

By R. F. Clippinger

COBOL was developed by the COBOL Committee, which is a subcommittee of a larger committee called the "Conference on Data System Languages" which has an Executive Committee and, in addition to the COBOL Committee, a Development Committee. Within the Development Committee are two working groups called the Language Structure Group (whose members are Roy Goldfinger of IBM, Robert Bosak of SDC, Carey Dobbs of Sperry Rand, Renee Jasper of Navy Management Office, William Keating of National Cash Register, George Kendrick of General Electric, and Jack Porter of Mitre Corporation), and the Systems Group. Both of these groups are interested in going beyond COBOL to further simplify the work of problem preparation. The Systems Group has concentrated on an approach which represents a generalization of TABSOL in GE's GECOM. The driving force behind this effort is Burt Grad.

The Language Structure Group has taken a different approach. It notes that FACT, COBOL, and other current data-processing compilers are "procedure-oriented," that is, they require a solution to the data-processing problem to be worked out in detail, and they are geared to the magnetic-tape computer and assume that information is available in files on magnetic tape, sorted according to some keys. The Language Structure Group felt that an information algebra could be defined which would permit the definition of a data-processing problem without postulating a solution. It felt that if this point were reached, then certain classes of problems could be handled by compilers that would, in effect, invent the necessary procedures for the problem solution. One sees a trend in this direction in FLOW-MATIC, COBOL, Commercial Translator, and FACT, if one notes that a sort can be specified in terms of the input file and the output file, with no discussion of the technique required to create strings of ordered items and merge these strings. Similarly, a report writer specifies where the information is to come from and how it is to be arranged on the printed page, and the compiler generates a program which accomplishes the purpose without the programmer having to supply the details. The FACT update verb specifies a couple of input files and criteria for matching, and FACT generates a program which does the housekeeping of reading records from both files, matching them and going to appropriate routines depending on where there is a match, an extra item, a missing item, etc. A careful study of existing compilers will reveal many areas where the compiler supplies a substantial program as a result of having been given some information about data and interrelationships between the data. The Language Structure Group has, by no means, provided a technique of solving problems once they are stated in abstract form. Indeed, it is clear that this problem is not soluble in this much generality. All the Language Structure Group claims is to provide an information algebra which may serve as a stimulus to the development of compilers