RAPIDWRITE—A New Approach to COBOL Readability

By E. Humby

Features in an autocode which make programs easier to read are often just those which make it irksome to write. The way to avoid an ugly compromise is to make Readability not something which affects the design of the language but a feature which can be added during the translation run. This paper describes I.C.T. RAPIDWRITE, an autocode enabling COBOL facilities to be expressed in condensed form from which is produced, at translation time, a full and valid COBOL version with its advantages of Readability and Compatibility.

The Readability Problem

A good autocode has to meet several requirements. It should be easy to learn, easy to use, and should produce programs which are easy to read, easy to write, easy to get working, and easy to modify. Different users in differing circumstances put different emphasis on these several requirements, and it is for this reason that no single autocode has universal acceptance. The effects of some of the requirements on the design of a language may be incompatible. In particular it is often the facilities which make a program easier to read by the uninitiated that makes the autocode less attractive to the person who has to write the program down. Some of the strongest criticisms of COBOL stem from programmers, who view with distaste the need, after having thought out the problem in broad steps, to write each step down in a long English sentence. On the other hand, many connected with the original design of COBOL have placed ceaseless emphasis on the requirement of Readability, at the cost to the writer of long data-names and constant repetition of the 15 verb formats. In COBOL 61 "IS GREATER THAN" constitutes "required" COBOL but the symbol ">" is "elective." The ADD, SUBTRACT, MULTIPLY & DIVIDE verbs are "required" but "COMPUTE formula" is "elective." I have yet to meet the accountant who whilst understanding

MULTIPLY RATE AND HOURS GIVING BASIC.
ADD BASIC AND BONUS GIVING GROSS.

could not understand

COMPUTE GROSS = RATE * HOURS + BONUS.

It is a great pity that many people have thumped the table on Readable autocodes, that many others have rejected COBOL simply because of its verbosity, and yet again that many have wasted time in trying to formulate compromises in the design stage. A great pity because there is a neat solution to the problem which retains all that is best in both worlds.

The General Solution

The escape from an apparent impasse stems from the fact that Readability is required, in the main, only after the program is complete. It need not colour the design of the autocode therefore, provided it can be added at a later and more convenient time—ideally at translation time. Having accepted this, the language in which the programmer writes can be condensed to the minimum which he requires. He will require only that separate sections of the program can be identified by mnemonic paragraph names, and his data-names may be abbreviated to any extent so long as he personally can remember what each means. He will not need to use any words or signs which are not significant to the description of the problem, and which might cause him trouble at testing time because of mis-spelling or incorrect punctuation.

The Rapidwrite Solution

I.C.T. *Rapidwrite* was designed with this end in view: of being able to write programs for a COBOL processor without imposing the burdens of readability on the writer. I.C.T. are convinced that COBOL is a valuable tool for business users of computers, and are offering a COBOL translator with its 1301 and later computer configurations. *Rapidwrite* is an alternative form of input to this processor. At the same time as making the language easier to write it is made easier to learn by omitting some of the COBOL facilities with marginal utilities, and also made easier to use by the provision of preprinted documents so that the available facilities are clearly displayed and the writer guided to the requirements for completion.

Examples: *Environment.* A typical COBOL Environment statement would be

OBJECT-COMPUTER 1301, MEMORY SIZE 400
I.A.S. WORDS 12000 DRUM WORDS.

Since the computer number is implicit in the program, the only items of significance to the compiler are the "400" and the "12000." The *Rapidwrite* Environment form is designed so that there are two obvious boxes in which only these two capacities have to be written. Similarly boxes are provided against names of the input output devices. The file identifier letter "T," for example, written in the appropriate box suffices for the full COBOL expression

FILE CONTROL. SELECT COMMODITY-TOTALS ASSIGN TO PRINTER.

Data Division

For Data description a tabular form is provided. There are "Redefines," "Occurs," and "Value" columns

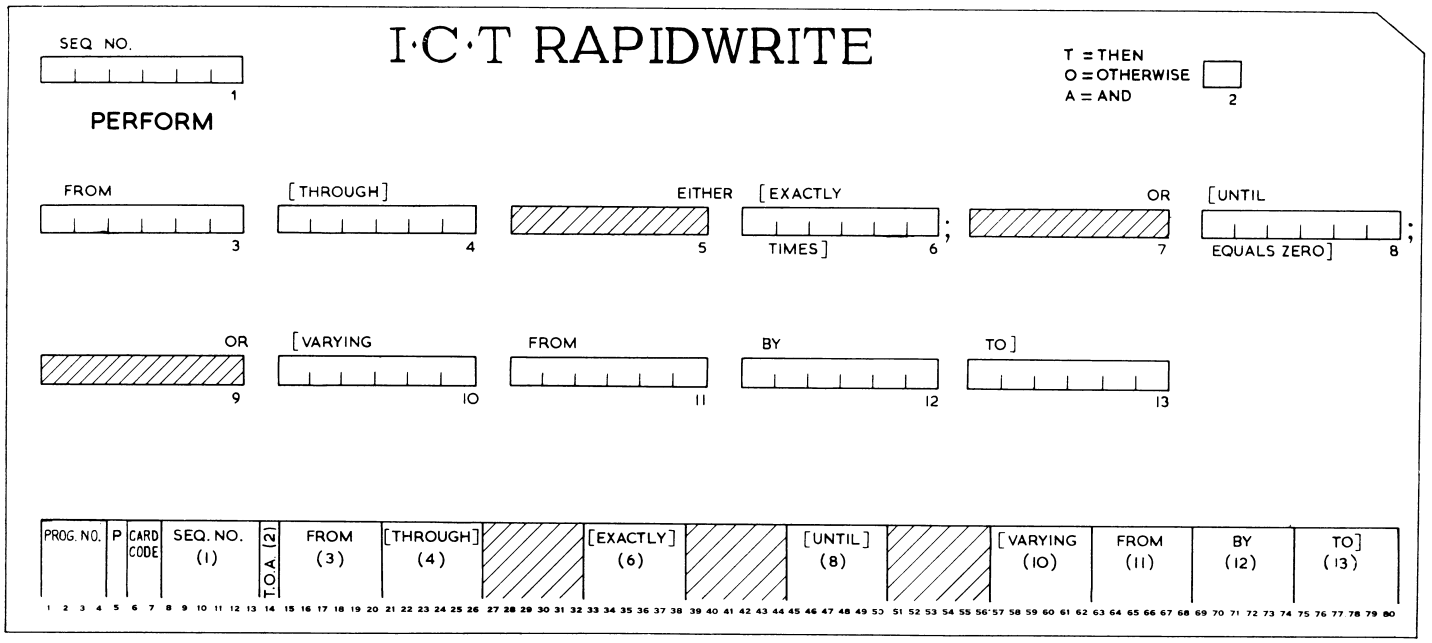

Fig. 1.—Example Rapidwrite card—Perform

and, apart from these, everything else to be defined against each name is coded in the "PICTURE" column. This entails the addition of some extra codes, but in this we seemed to have partly anticipated COBOL 61. A *Rapidwrite* picture: T999(9V99) contains all the information given by the COBOL statement

> SIZE IS 6 CHARACTERS, SIGNED, POINT LOCATION IS LEFT 2 PLACES, CLASS IS NUMERIC, ZERO SUPPRESS LEAVING 3 PLACES.

and a tabulation of such pictures is much more descriptive of a record content than a sequence of COBOL statements.

Procedure Cards

In the Procedure Division, since COBOL is a restricted form of English, certain sentence format groups keep repeating, and only the data-names involved change from occasion to occasion. *Rapidwrite* allows 11 different statement formats: READ, WRITE, COMPUTE, MOVE, IF, GO, STOP, PERFORM, INCLUDE, SUBSCRIPT & PARAGRAPH-NAME. The constant parts of these statements are pre-printed on cards, and boxes are provided at the appropriate points in which the programmer writes the data-names, literals, etc., which are involved on each occasion. (Fig. 1 shows the PERFORM CARD as an example.) The programmer selects and completes the procedure cards one by one, from a tub file, as he interprets the basic flow chart. This unit method of constructing the procedure enables the statement cards to be laid out as they are prepared, as though they were blocks in a block diagram. In this form the redirecting of jumps, the revision and insertion of pieces of program are considerably simplified. The number of rubbers consumed and waste-paper baskets filled is remarkably lower during the programming phase.

Some idea of the volume of writing saved is given by Figs. 2 and 3. Fig. 2 shows part of a program in COBOL. Fig. 3 shows the amount of writing for a *Rapidwrite* version of the same problem. It should be remembered that this is much more comprehensible when it appears on the preprinted sheets and cards.

The Processor

The significant information shown is punched from the Environment and Data sheets and into the dual-purpose Procedure cards; the resulting pack constitutes the input to the translation process which produces the machine-code version of the program. Fig. 4 shows broadly the process of translation. When the processor is accepting a full COBOL program via entry 2, the main task of Phase 1 is to extract the significant parts of the sentences, but it will incidentally print a copy of the input program. A *Rapidwrite* program would be accepted by entry 1. This time the program is almost already reduced to its elemental significances, but the valuable contribution of phase 1 is to dress up the bare bones of the *Rapidwrite* statement into a readable and valid COBOL printout. The necessary format words are ready stored in a dictionary. In addition a data-name synonym dictionary can be loaded as well. For example, in an earlier example the programmer used "T" as a file-name, but the systems-analyst may insist on the use of "COMMODITY-TOTALS." If in the synonym table is entered

T = COMMODITY-TOTALS

the COBOL printout in phase 1 would contain the long name wherever the programmer had written the short name.

```
SEQ. NO.   NARRATIVE
           1 START. OPEN INPUT ISSUES OUTPUT SUMMARIES, COMMODITY-TOTALS.
           2        READ ISSUES AT END GO TO ERROR-1.
           3        MOVE CUSTOMER-NO OF ISSUES TO CUSTOMER-NO OF SUMMARIES.
           4 CALCULATION. COMPUTE NET-VALUE = NET-VALUE + (GROSS-VALUE OF ISSUES -
           5        GROSS-VALUE OF ISSUES * DISCOUNT/100).
           6        ADD QUANTITY OF ISSUES AND STORED-QUANTITY(STORES-NO OF ISSUES).
           7        ADD GROSS-VALUE OF ISSUES AND STORED-GROSS(STORES-NO OF ISSUES).
           8        READ ISSUES AT END GO TO PRINTING.
           9        IF CUSTOMER-NO OF ISSUES IS LESS THAN CUSTOMER-NO OF SUMMARIES
          10        STOP "WRONG SEQUENCE".
          11        IF CUSTOMER-NO OF ISSUES IS GREATER THAN CUSTOMER-NO OF SUMMARIES
          12        PERFORM CUSTOMER-CHANGE.
          13        GO TO CALCULATION.
          14
          15 CUSTOMER-CHANGE. ADD NET-VALUE AND TOTAL-NET-VALUE.
          16        WRITE CUSTOMER-SUMMARY.
          17        MOVE CUSTOMER-NO OF ISSUES TO CUSTOMER-NO OF SUMMARIES.
          18        MOVE ZEROS TO NET-VALUE.
          19
          20 ERROR-1. STOP "NO TRANSACTIONS".
          21
          22 PRINTING. MOVE TOTAL-NET-VALUE TO NET-VALUE.
          23        MOVE ZEROS TO CUSTOMER-NO OF SUMMARIES.
          24        WRITE CUSTOMER-SUMMARY.
          25        PERFORM TOTAL-LINE-PRINT VARYING NO FROM 1 BY 1 TO 999.
          26        MOVE TOTAL-GROSS TO GROSS-VALUE OF COMMODITY-TOTALS.
          27        MOVE ZEROS TO STORES-NO IN COMMODITY-TOTALS AND QUANTITY IN
          28        COMMODITY-TOTALS.
          29        WRITE TOTAL-LINE AFTER ADVANCING 4 LINES.
          30        CLOSE ISSUES, SUMMARIES, COMMODITY-TOTALS AND STOP RUN.
          31
          32 TOTAL-LINE-PRINT. IF STORED-GROSS(NO) IS NOT ZERO OR STORED-QUANTITY
          33        (NO) IS NOT ZERO THEN MOVE NO TO STORES-NO IN COMMODITY-TOTALS,
          34        MOVE STORED-QUANTITY(NO) TO QUANTITY OF COMMODITY-TOTALS,
          35        MOVE STORED-GROSS(NO) TO GROSS-VALUE OF COMMODITY-TOTALS,
          36        ADD STORED-GROSS(NO) AND TOTAL-GROSS,
          37        WRITE TOTAL-LINE.
```

Fig. 2.—Part of a Program written in COBOL

```
  1    PA START
  2    RE I      ERROR
  3    MO CU-NOICU-NOS
  4    PA CALC
  5    CO NET . NET +(GROSS-GROSS*DISC/
  6 E  CO 100)
  7    CO STQTY STQTY + QTY
  8    SU ST-NO ST-NO
  9    CO STGRS STGRS + GROSS
 10    SU ST-NO ST-NO
 11    RE I      STATS
 12    IF CU-NOI <    CU-NOS
 13 T  ST CU-NOS
 14    IF CU-NOI >    CU-NOS
 15 T  PE CU-CHG
 16    GO CALC
 17    PA ERROR
 18    ST "NIL"
 19    PA CU-CHG
 20    CO TONET TONET + NET
 21    WR CU-SM
 22    MO O  F  NET
 23    MO CU-NOI      CU-NOS
 24    PA STATS
 25    MO O  F  CU-NOS
 26    MO TONET NET
 27    WR CU-SM
 28    PE P-LINE      NO 1 1 999
 29    MO O  F  P-SNO P-QTY
 30    MO TOGRS P-GRS
 31    WR TOTLN  A 4
 32    ST "RUN"
 33    PA P-LINE
 34    IF STQTY NOT= 0  OR STGRS NOT= 0
 35    SU NO                  NO
 36 T  MO NO    P-SNO
 37 A  MO STQTY P-QTY
 38    SU NO
 39 A  MO STGRS P-GRS
 40    SU NO
 41 A  CO TOGRS TOGRS + STGRS
 42    SU                NO
 43 A  WR TOTLN
```

Fig. 3.—This is all that is written and punched in the RAPIDWRITE version of the piece of program given in Fig. 2

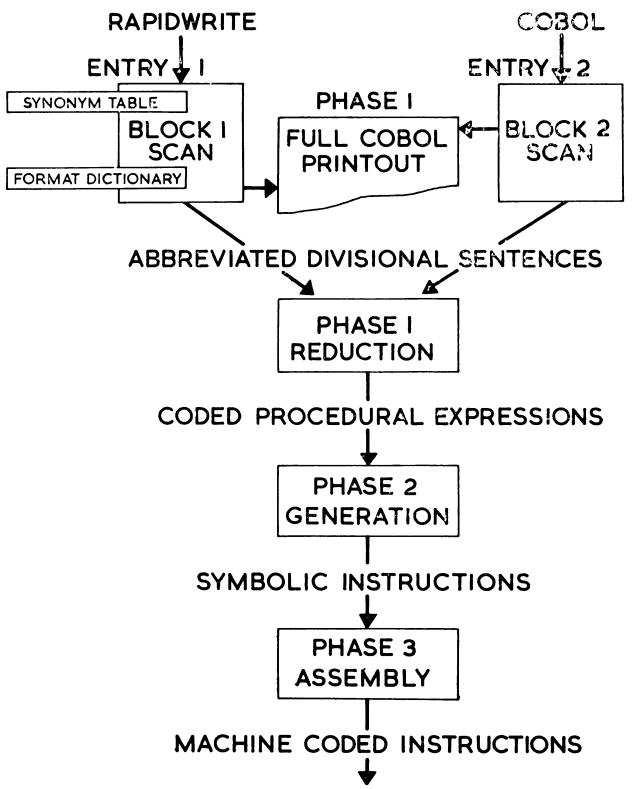

Fig. 4.—The Processor

Another Kind of Translation

As well as helping the programmer who wants to use a language that is more economic than COBOL English, *Rapidwrite* comes to the aid of programmers who do not understand English. The basic sentences on the preprinted *Rapidwrite* cards might equally well appear in French or German. Moreover, provided they are made up from the set A to Z, 0–9 and hyphen, data-names may be invented that are mnemonic to the program writer in his own language. It requires only the replacement of the format dictionary table by a German one to give a printout at translation time which is readable to someone who understands German only. This kind of flexibility is worth thinking about. The synonym table could be used not only for the substitution of long English names for short English names, but could also be used, for example, to substitute long English names for short Italian names. This would mean that an Italian organization where no English was spoken could enjoy the benefits of COBOL and, furthermore, it would only require one run using different dictionary and synonym tables to provide a complete English COBOL version of any of their programs, thus providing Readability and Compatibility over a much wider area. Is this not making COBOL truly international?

Conclusions

Anyone who has examined COBOL and liked it, and anyone who has examined COBOL and disliked it, should give it a fresh appraisal considering I.C.T. *Rapidwrite* as an alternative means of expression. I.C.T. *Rapidwrite* allows the programmer to write in a fashion convenient to him but provides automatically COBOL Compatibility and Readability even across natural language boundaries.

Bibliography

"COBOL 60," U.S. Dept. of Defense. (April 1960)
"Rapidwrite Programming Manual," I.C.T. Ltd. (Sept. 1961.)

Book Review

Automatic Control and Computer Engineering. Edited by V. V. Solodovnikov, translated 1961 (Oxford: Pergamon Press Ltd., £5).

This volume consists of translations of 24 assorted papers presented at a session of the Academy of Sciences of the USSR devoted to examining the problem of complete complex automation of manufacturing processes. We are, unfortunately left to guess just when this session was held, but since the original Russian volume was published in 1958 and references up to 1957 are included in some papers, the meetings were presumably held early in 1958. This means that all the information contained in the book is somewhat dated, since the fields which the book covers are at a stage of rapid development, and three years have seen tremendous advances. Also, since the I.F.A.C. Congress held in Moscow in June 1960 contained many papers on the automation aspects covered in the present volume, workers in the field are quite well informed on Russian developments.

One interesting point arises from the references at the end of the paper by Kopai-Gora. The first seven refer to a report at an "All-Union Conference on the automation of manufacturing processes, Magnitigorsk (May–June 1948)." If this is the correct date it is quite remarkable that some of the present authors were able to give papers with almost identical titles ten years beforehand. If, on the other hand, the date is a misprint for 1958 then the date of the conference is fixed and we are left to wonder whether these authors wrote more than one similar paper for the conference, or whether we have been presented with different renderings of the same titles.

This raises a most serious criticism of the book. The translation in many places is of a most literal nature, presumably caused by a lack of familiarity with the subject on the part of the translators. The paper by Mamonov entitled "The Use of Semiconductor Instruments in Computer Engineering" is about the use of transistors and crystal diodes. I suspect that the word translated (correctly) as "Instrument" is the Russian "pribor" which can also correctly be translated as "device" which would be more satisfactory. This is only a simple example. On pages 226 and 227 there are figures showing the "Registering numerical material from —" instead of "reading numerical data from —" and on pages 113, etc., we have a long dissertation on "Summators" using "cadence pulses." However, this same paper also shows the reverse error which occurs throughout the paper by Zimin on logical circuits where we are introduced to the ILI, I, and NYeT circuits. These are transliterations of the Russian words for "or," "and," and "not" which, of course, are the normal terms for these circuits. There are many other such cases, and in fact I wonder if it is a waste of human endeavour to produce translations of this nature. Surely this translation suffers from exactly those faults which are supposed to beset present-day machine translation, and if this is so I would prefer the imperfect machine translation at an earlier date and possibly at a lower price.

Returning to the subject matter of the papers, it must be pointed out that there are interesting sections such as the group of papers on steel automation, but a large proportion of the book covers elementary facts in great detail. It is odd, and yet characteristic of Russian papers, that constructive work is often hidden amongst a mass of fundamental detail which might, translated properly, form a sound text-book.

On the whole I cannot see many individuals wishing to spend £5 and "500 pages" of reading time for the rewards contained in the book, but it may well be a useful acquisition for libraries as a background work.

LAURENCE CLARKE.