User's experience of COBOL

By I. M. Golds*

This paper, and the two following ones by T. H. Ayre and M. Richardson, were presented at the opening session of the symposium on *Practical Experience with Commercial Autocodes* held in London on 25 March 1964 by the Advanced Programming Study Group of the B.C.S. These three papers briefly describe the experience, with commercial automatic programming systems, obtained by computer users in three Government departments.

The programs

The IBM 705 Mark 2 computer used for the production of results from the 1961 Census of Population has magnetic tape input and output and a core-storage of 40,000 character locations, using a binary-coded decimal representation with variable-length words but a fixed length for instructions; a card-reader is available on-line at the installation. Programs for the major part of the work, including the tabulation of figures for publication in County reports, were written in a machine-oriented language—Autocoder—which was basically a system of one-for-one mnemonic codes for each machine instruction, with some simple macro-instruction facilities. In addition to the tabulations at county level, Census results are published in national reports on particular subjects. Tabulating the figures by computer for the national tables, which number about one-hundred-and-fifty, presented a problem to the programming team. When tested the programs would only be used for two production runs, giving a very small return for the time spent on them by the programmers. There was clearly a need to keep the time required for writing and testing the programs to a minimum, and COBOL 61 was introduced in the hope of achieving this. Since the programs were only to be run twice it was not of primary importance that the object programs should require the optimum amount of running time; the similar county tabulations required on average about half the available memory space in the computer, therefore economic use of core-memory was not a critical factor.

In general the tabulation programs have single tape input, the records consisting of the basic counts of figures needed for a substantial number of tables. Output is also written on to a single tape, in card-image records containing the figures for one table in report format, together with codes for controlling the action of the IBM cardatype machine which is used to print out the tables from the cards when punched. The figures may be obtained directly from the Input records or derived by calculations.

A team numbering three to four programmers has been writing these programs over the past twelve months. Programs for about forty tables have so far been compiled; they consist on average of 200–250 statements in the Procedure Division.

Adequacy of programming facilities

The compiler provided by IBM makes use of an existing, more advanced version of the Autocoder assembly program, translating the COBOL source-program into Autocoder macro-instructions as an intermediate stage. It has one notable limitation for us, namely that it does not provide for the translation of COBOL entries referring to tape input or output for the mark 2 version of the 705; we have therefore incorporated our own existing and tried Autocoder Input-Output routines. Because Autocoder is so much involved, and in the absence of a satisfactory system of source-program debugging, a knowledge of Autocoder has been essential for the programmers.

Some of the editing features have also not been available to us; one of these in particular—namely a floating minus sign at the left end of a numeric field—would have been very useful. We would also have liked a facility for placing a dash in numeric fields when the value is zero, but this is unlikely to be provided in any version of COBOL.

Many of the features provided have been particularly useful in the tabulation programs. Much use has been made of the subscripting facility in conjunction with the "occurs" clause, especially in referring to the series of counts in the input records. The "move corresponding" option and some of the available editing features have been helpful in setting up output records. The "picture" clause has been used throughout the Data Divisions because it provides a very necessary shorthand method of describing the attributes of data-items. The "compute" verb is another useful form of shorthand. Finally, of course, we have been very glad of the provision for entering another language.

Training

A formal course which was given to the programmers lasted 9 mornings; all the programmers attending this had some varying amount of previous experience in writing and testing Autocoder programs. In general, after the course the programmers had a working grasp of COBOL and were able to produce satisfactory programs straight away, speeding up after completing their first.

* *General Register Office, c/o The Royal Army Pay Corps Computer Centre, Worthy Down, Near Winchester, Hants.*

Writing source programs

It has been difficult to assess how much time COBOL has saved at the program-writing stage. If flow-charting is carried out in sufficient detail to cope with all the logic involved, coding the Procedure Division takes noticeably less time than with Autocoder; but this means that time spent on the flow-chart is as long. Writing the data definition entries takes some time, principally because there is always a substantial amount of redefining to be done to enable the data to be handled efficiently in all parts of the Procedure Division. Saving in writing time has been achieved, then, only at the stage of coding the program steps. However, COBOL source programs are very easy to read and this aids the desk-checking operation considerably; it also means that there is no need to write a description of the action of the program when completing the documentation.

There is one drawback at the program-writing stage which should be mentioned here. It is possible to make errors in COBOL entries by offending against some of the many conventions of syntax and punctuation without being aware of it; some compound conditional expressions are a particular source of confusion in this respect.

Compiling the programs

The compiler is constructed to process programs in two stages. In the first it translates the COBOL paragraphs into series of Autocoder macro-instructions, and also lists diagnostic error messages; this run is repeated (usually once) until the messages are cleared. The second stage is a modification of the Autocoder assembly run, expanding the macro-instructions and assigning memory locations. We had to write our own program to make the output of the first stage acceptable as input to the Assembly stage, as there was otherwise no provision for this and the first stage had to be run again unnecessarily once the program was corrected, in order to lead into the second part. This modification we have made to the compiling procedure has resulted in some trouble with unrecognized operands, and in any case means that the final printed output is in two listings.

Computer time needed for compiling has proved to be ten times that required to assemble similar programs in the basic version of Autocoder, but part of this expansion factor is accounted for by the time taken by the more high-powered version of Autocoder used. A tabulation program taking 20 minutes for a basic Autocoder Assembly would take 1 hour 20 minutes assembly time using the more advanced form. This, coupled with 1 hour taken up by each run of the first (translation and diagnosis) stage of compilation, would result in 3 hours 20 minutes being required to compile an average program.

With all the various options which the compiler must allow for, we are still finding it possible to compose entries which it fails to cope with correctly, and even some which reveal restrictions we had not heard of. About twenty-five per cent of our object programs have contained some minor compiling error, and there has been a steady stream of corrections to the compiler from IBM.

Testing

The procedure used for testing relies on a series of utility programs and provides for applying corrections to the object program from patches punched into cards, using machine-codes, and for obtaining a print-out of memory and output-tape records at the end of a test. The prints are examined in conjunction with the listings of the source- and object-programs.

Testing time has been significantly reduced; quite a number of programs have required only three or even two tests, although most have taken more than that. Even ten tests would be a good saving compared with the number we needed for Autocoder programs. There are two reasons for the improved testing time: COBOL virtually eliminates all errors in the object program other than those of logic or results of the programmer misunderstanding a convention, or possibly of a compiler fault. Also, at least with the compiler which we have experienced, it is conducive to error-free patching because the set of machine instructions generated for each COBOL statement is self-contained, and this reduces the likelihood of patches relating to one routine affecting another.

Object programs

The object programs have two major defects, which were expected: they are uneconomic of memory space, and running time is prolonged. Some of the sequences of machine instructions have contained completely superfluous steps. Varying subscript data-names is particularly inefficient, as routines to calculate subscripted addresses are sizeable and have to be executed for each variation of the subscript; a facility for direct address modification, instead, would save time and memory space.

Even with COBOL, intelligent programming can have an effect on the time taken to run object programs.

Comparing a county tabulation program written in simple Autocoder, which filled half of core-memory, with the COBOL program for the equivalent national table which was written by the same programmer, the expansion factor appears to be about one third, So far only two of our programs have had to be clipped in order to fit the capacity of the computer, but if COBOL were more economic of space we could have made more of our programs cope with more than one table and thus saved computer time and even some programmers' time.

Conclusion

Considerations of memory space and computer time would have been much more decisive factors in applying COBOL to the programs for the earlier stages of the Census operation. The principal aim in introducing it

for the national tabulations, namely to save programmers' time, has been achieved. The programmers themselves find it cuts much of the tedium out of their work, and they are now trained in a programming language which could be used for a later Census even when a change of computer is necessary. The programs could also be used in the future without being extensively rewritten, and the ease with which they can be read should be helpful when any amendments have to be made.

User's experience of RAPIDWRITE

By T. H. Ayre*

The Ministry of Public Building and Works uses an I.C.T 1301 with 2,000 words of IAS and 24,000 words of drum storage, to which are fitted eight one-inch magnetic-tape units operating at 90,000 c.p.s. The work already being done or in an advanced stage of preparation includes payroll, stock-control, payment of accounts and the processing of statistical returns from the building industry. In all, some 40 or 50 different regular runs are or will shortly be in use, and the total amount of program is accordingly considerable.

RAPIDWRITE

ICT RAPIDWRITE is closely based on COBOL, the major differences being that the procedure division is written and punched on dual-purpose cards, that names are restricted in length to five characters and that, in the version available when we were using the language, no facilities were offered for processing magnetic tape.

Ease of use

In our experience the degree of ability required to produce satisfactory programs is as great in RAPIDWRITE as in machine language. The writing time also seems to be comparable for both methods. This is to some extent due to the absence of tape facilities, which necessitates the writing of considerable amounts of machine-language program before any RAPIDWRITE-based program is usable.

For data divisions of any complexity the restriction on the length of data names enforces the use of extreme abbreviations, which are often as meaningless to the uninitiated as would be machine-language addresses. The absence in RAPIDWRITE, as in COBOL, of facilities for sorting, merging or searching files would in any event severely restrict the usefulness of these languages for commercial purposes.

The facility for specifying the content of a print line in plain language, and having the line arranged and printed without further effort would be of great assistance if it were possible to isolate it from the less desirable features, especially as there are no built-in aids to the programmer for this purpose.

The compiler which we have is designed to operate on a non-tape 1301 with only 400 words of IAS, and is wasteful of computer capacity when compiling is done on a much larger computer. Compiling time is of the order of three to four hours for average programs, even after we have modified the compiler to produce its object program direct to magnetic tape instead of in the form of punched cards.

The printed object program produced is in pure machine language, and indeed is of necessity relativised in a form somewhat more involved than a machine-language programmer would willingly use.

Debugging of logical errors can be done in RAPIDWRITE only at the cost of a full recompilation and the "refitting" of all the machine-language tape routines. For this reason all debugging is done by machine-language patching of the object program, and all programmers therefore need an adequate knowledge of machine-language methods.

Ease of debugging

It is undoubtedly true that the absence of clerical errors and storage misallocations in compiled program reduces the number of proving runs necessary with a RAPIDWRITE program. On the other hand the difficulty of following the compiled interpretation of the source program may mean that a longer time elapses between one proving run and the next. On balance the first factor seems to outweigh the second, so that a fully-proved program may be expected appreciably sooner after the first test of a RAPIDWRITE program than would be the case with a comparable machine-language program.

Use of storage

The total volume of program produced by RAPIDWRITE is inevitably greater than that of a program written directly in machine-language. The only direct comparison we have been able to make showed a use of about 6,000 words of storage for RAPIDWRITE as against about 1,500 words for machine language. The job in question is described in the next Section.

* *Ministry of Public Building and Works, Accounts Division, Lambeth Bridge House, London, S.E.*1.