Running time

In considering the facts about to be quoted it is important to remember that the version of RAPIDWRITE we used was intended to produce object program for a non-tape 1301 with only 400 words of IAS. Any IAS in excess of this may be used only for data areas. For this reason most of the running time is occupied in transfers of fresh supplies of program from the drum. In the case of input and output operations the large number of such drum transfers necessary for each line or card reduces the running speed to as little as one quarter of the maximum speed of the printer or card reader.

A machine-language program with the same storage limitations as the compiler might also fall short of maximum speeds, and to this extent RAPIDWRITE might compare less unfavourably with machine language on such a computer.

A monthly task is to mark off some 2,000 payable orders cashed during the month against a tape file containing, amongst other details, particulars of all payable orders issued. The file occupies slightly more than one 3,600 feet of tape, the full-speed passing time of which would be about 6 minutes. A balance sheet is accumulated during the run and printed out in about a dozen lines at the end.

This program was written in RAPIDWRITE by a programmer who was familiar also with machine language. When the program was complete and proved it was allowed to run for some two hours, and then an assessment was made of its progress. It was apparent that the full running time would be about eight hours. The original programmer then undertook an optimizing operation, consisting mainly of writing out some of the multitudinous drum transfers arising from use of the PERFORM option, and the job, reduced in time to about two and a half hours, was released for use.

This time was, of course, still much too slow for permanent use, and when it became possible to divert a programmer it was rewritten throughout in machine language, using the systems work already done and working from the excellent flow charts produced for the RAPIDWRITE program. The rewrite was proved on its third test, and runs in 8½ minutes.

User's experience of CLEO

By M. Richardson*

The Board of Trade Census Office has used CLEO for preparing Annual Inquiry registers, and issuing and processing the 1963 returns. We have also used CLEO for a register of new companies and are now in course of applying it to all new jobs. Comments made here relate to the first seven programs written, which were analyzed in some detail.

The first CLEO compiler did not become available until the end of 1963 and is still not on general release.

1. General comments

(a) Adequacy

So far CLEO has covered all our programming requirements since these needed only integer arithmetic. It has not been found necessary to enter the lower language, i.e. Intercode. The major extension of facilities to floating-point values, etc., will cover all future requirements; there will be no need to know Intercode.

Facilities are to be extended to cover function calls and floating-point working within the next three months. In general, most of the features mentioned in T. R. Thompson's paper in this *Journal* have been included in the system or will be included very shortly. At present the major limitation is on editing values for printing; this has been covered by a CLEO subroutine.

(b) Ease of learning, etc.

The CLEO conversion course provided for the first team of Intercode programmers occupied one and a half days. There is no doubt that CLEO is easy to learn and with the compiler checks provided it is easy to use correctly.

(c) Debugging

Amendments are very simple to make in CLEO. The trials data system now being provided is even simpler than the Intercode system. It was difficult to locate logical errors in Intercode, but it is easy to do this in CLEO. Desk checking becomes more effective.

(d) Compiler problems

Only one minor imperfection appeared in the first version; this is now being put right.

2. Satisfactory features

The following table gives some idea of the reduction in programming time made possible by CLEO.

	RELATIVE PROGRAMMING EFFORT (time units)	
	INTERCODE	CLEO
Flow charting	10	0
Coding time	55	10
Data description	10	2
Checking	10	2
Proving	15	2

* *Board of Trade, Census Office, Lime Grove, Eastcote, Ruislip, Middlesex.*

Reduction in time between specification and final proving run may be as much as 75%.

Documentation has been helped enormously and standardized data descriptions, etc., have also contributed to economies achieved under the new system.

3. Unsatisfactory features

(i) *Expansion of files*

An extra five characters are added to every record. In the case of small records this could be important, and extra steps may be needed to create block-records in these cases.

(ii) *Incompatibility of files*

Intercode files cannot be read directly by CLEO programs.

(iii) *Compiling timc*

In general this may be equal to the normal translation time which is an integral part of the compilation process. In other words, the total translation time is nearly doubled. For average programs this time increases from 8 to 16 minutes. Savings are achieved, however, by substantial reduction in the number of re-translations and amendments required.

(iv) *Inefficiency*

Program expansion seems to be of the order of 10%, but efforts are being made to reduce this. The main inefficiency is related to the file expansion referred to above.

(v) *Training*

There is still no Teach-Yourself-CLEO manual.

(vi) *General*

We are in no way critical of the compiler's editing and reporting action. This is even better than we had expected. Our chief criticism is of the compiler's late arrival.

4. Conclusions

All our new programs will be written in CLEO, and all existing files will be converted to CLEO format. We have found that this system helps to balance programming and computing capacity, and furnishes a basis for training systems-analysts.

Reference

Thompson, T. R. (1962). "Fundamental principles of expressing a procedure for a computer application," *The Computer Journal*, Vol. 5, p. 164.

Book review: Mathematical programming

Recent Advances in Mathematical Programming, Edited by Robert L. Graves and Philip Wolfe, 1963; 347 pages. (Maidenhead: *McGraw-Hill Publishing Company Ltd.*, 92s.)

It is dangerous to include words like "recent" into the title of such a collection as this, which contains contributions to a Symposium held in 1962 at the University of Chicago. What was new then, is now at best a basis on which to build even more recent developments. Such thoughts have some topical significance in 1964, since in July of this year the sequel to the Chicago meeting will be held in London, sponsored by the *British Computer Society* in common with other associations.

The 43 titles in the book under review refer to 23 full papers and 20 abstracts, concerned with theory, applications, and computational aspects. We may assume that it is the last-mentioned category about which readers will expect to find some comments in this *Journal*.

Most papers on computational methods refer to the Simplex Method of Linear Programming, or to its adaptations to the non-linear case. M. Balinski describes a method for solving pairs of dual programs, while P. Wolfe gives a survey of non-linear methods, with a bibliography. Clair E. Miller describes a method for separable programming (the objective function consists of additive terms each of one single variable), which has proved to be applicable to many problems which arise in practice. G. B. Dantzig describes a "compact basis triangularization," and E. M. L. Beale the use of pseudo-basic variables, a method which is, in a sense, the dual to the Dantzig-Wolfe decomposition principle. Other algorithms inspired by the latter are given by J. M. Abadie and A. C. Williams, and by J. B. Rosen.

P. Wolfe and Leola Cutler report on tentative conclusions from SCEMP (Standardized Computational Experiments in Mathematical Programming) and D. M. Smith and W. Orchard-Hays on experiments with product form codes. Glenn T. Martin describes an accelerated algorithm for integer programming.

It will, of course, be appreciated that the practical man can also derive useful ideas from theoretical papers, and even from abstracts. It is therefore recommended that he should at least glance at the whole collection.

In any case, whoever the reader, the book is a useful reference to one stage in the development of mathematical programming, though perhaps not a very exciting one.

S. Vajda.