The gradual acceptance of a variety of commercial English languages

By R. M. Paine*

This paper is based on a talk given at the I.F.I.P. Congress, New York, May 1965, a summary of which will appear in Vol. 2 of the *Proceedings* of that Congress.

In Britain, computer users have not rushed to embrace automatic coding for business applications, in fact they have been in the past rather hesitant to try it at all. There have been many programmers present at lectures on the subject—but the realistic chief accountants, or sceptical managers of the information services department, have been reluctant to decide that all programs in their computer department will be written in automatic coding. It is fashionable to know about the newest languages, but it is not so accepted actually to use them on real business work—that is regularly repetitive runs.

Short history of developments

What are the reasons for this hesitancy? The normal well-known British reserve and suspicion of anything new plays its part—even among the computer profession, who are considered innovators by other professions. Also, perhaps, the fact that autocoding languages—especially COBOL—spread from America, created some native resistance. This resistance was in spite of, or perhaps even because of, the fact that about 40% of the commercial computers installed in Britain were American. British computer manufacturers had been slow to provide assembly languages, let alone full automatic-coding languages—although much early work and many conjectures about autocodes, mainly scientific, had taken place in the United Kingdom. (Brooker 1958; Clarke and Felton 1959; Gill 1959.)

The machines installed in Britain—at least up to 1960–62—whether British or American in design, were smaller on the whole than those sold in America, so that there was less scope for higher level languages than in America. The British user had to cope with quite small stores, sometimes two-level stores, and fairly slow speeds, and quite often no magnetic tape—in these circumstances, the possibilities of automatic programming were somewhat limited (Conference Discussion 1962). In 1955, when I started working on computers, there were only about fifteen computers in the country. The fifteen or so computers of 1955 increased gradually to about 660 installed or on order by 1963; but in the last two years the pace has quickened considerably and there are now some 1,800 computers installed or on order (Miller, 1965). These figures include replacement machines, however, and may be too large for the actual number of computers that will be working. At least 1,150 of these, however, are in the 1401 class or smaller; and I need not tell you that COBOL on the IBM 1401 or the card I.C.T. 1301 is hardly an attractive proposition. So weaned on machine codes, the British users, even when bigger faster machines became available, were reluctant to go straight to languages such as COBOL. In many cases they accepted in the early 1960's the idea of symbolic languages, such as I.C.T.'s T.A.S., as a miraculous gift from heaven, though I believe American users had enjoyed them in practice for several years, and I remember using IBM's "SOAP" in 1958 (SOAP II, 1957).

Inefficiency

Also, quite rightly, the British user was concerned with the inefficiencies of automatic coding in regard to

(i) storage space;
(ii) object running speed;
(iii) the large amount of computer time required for compiling and de-bugging; and
(iv) the number of tape units required.

Alarming stories spread across the Atlantic about the inefficiencies of early COBOL and other languages (COBOL–60; Paine, 1960). In Britain, labour costs compared to machine costs are probably much lower than in America, so users were more prepared to use programmer's time rather than to use computer time in what they would consider a most wasteful way. The larger users, who might otherwise have thought of experimenting, were on the whole too busy with everyday bread-and-butter jobs such as invoicing, stock-control and payroll. Many users probably found they had been undersold in machine power and were struggling to save a machine instruction here and a constant there in order to get their program into storage, rather than facing the luxury of leaving a compiler to allocate storage, knowing they would have something in hand.

Compatibility and standards

British users were sceptical about compatibility right from the first communique issued by the Department of Defence on COBOL (COBOL–60). In Britain we did

* *Eastern Electricity Board, Ipswich.*

not think compatibility was possible, nor did we need it, since we did not have a massive investment in hardware and so were not faced with an enormous cost of reprogramming. Many people were not contemplating changing their machine, and those that did were in the main changing from a two-level card machine to a one-level magnetic-tape machine which required a change in system anyway. In contrast to the U.S.A., there was not a large IBM market to capture. Thus there was little drive to standardize on one language—in fact within the British Computer Society, the study group on automatic coding in the early 1960's was almost an anti-COBOL, anti-standardization group (Willey *et al.*, 1961a; Willey *et al.*, 1961b; Ellis, 1961). This lack of a standard or acceptance of COBOL was helped by there being several fairly small British manufacturers, who, as they got prodded into autocodes, developed a language for their own machine—they were not concerned whether such a language would run on other machines, or even on other models that they themselves produced. With so many manufacturers it was difficult to invest the large sums required to produce fully comprehensive software—the market for each model was too small really to recover the costs of the development of compilers (Paine, 1962; D'Agapeyeff, 1962). Thus many source languages appeared—some had only a short life—and there still are, as indicated in my title, a variety of commercial languages in Britain.

Current position

In the last couple of years things have changed—I might almost say "improved"—in the United Kingdom. Mergers among computer manufacturers have given greater financial resources to develop comprehensive software—including operating systems—and at the same time presented a larger market for a particular model which might be large enough to recoup the costs of software. Experience of writing earlier compilers has given manufacturers greater confidence in avoiding pitfalls, and they now advertise hard about the autocodes they can supply. Users have seen from America that performance of compilers has improved (Cowan, 1964), and at the same time, the size of store in use or on order in Britain has increased sufficiently to allow users to experiment, since 8K to 16K words are now fairly common. Many users now make it a condition of purchase that the manufacturer can supply an autocode, even if they have no intention of using one, since it is said to be the hallmark of an advanced manufacturer. Some manufacturers are also able to display their compilers working at almost the same time as they demonstrate their hardware, so users do not have to buy a pig in a poke and live on promises for the first two or three years of the installation. Some professional programmers have now advanced to the position of managers in their companies, and are showing themselves willing to make a far-sighted decision to use autocoding now, even if all its benefits will not be reaped until the next generation of compilers and machines.

So we have a gradual acceptance of autocoding—though still only a very small proportion of users write in autocodes (D'Agapeyeff, 1965). After that short history, I should now like to talk about the variety of languages offered and used—excluding those offered by American companies and concentrating on those supplied by British companies.

Mergers

Through mergers, the two largest British computer groups are International Computers and Tabulators, and English Electric—Leo—Marconi. I.C.T. was formed by successive mergers or take-overs of Powers–Samas, Hollerith, E.M.I. Electronics, and Ferranti—and I.C.T. also sell machines made by R.C.A. and Remington Rand. So you can see there is a fine mixture of derivations, languages and products in that collection—almost as rich a fusion as the development of the British or American nation itself.

English Electric—Leo—Marconi is not quite such a mixture. English Electric's Computer Department took over the computer interests of J. Lyons & Company Ltd. Lyons were the first firm in the world to have an electronic office, in February 1951, and they then produced and sold computers under the name of Leo* Computers. English Electric have lately moved their computer interests into their Marconi subsidiary, who also have had some computer developments, mainly military.

CLEO

Probably the commercial autocode most widely used in Britain today is CLEO, standing for Clear Language for Expressing Orders (Thompson, 1962; CLEO, 1961). This was developed by Leo Computers, for their Leo III and 326 computers, before their merger with English Electric. The Leo III and 326 are large, fast, time-sharing machines, and about fifty of them have been sold throughout the world. They can be compared with the Honeywell 800 or 1800 and the IBM 7040 or 7090.

CLEO was written specifically for Leo's own machines and did not seek to be a standard or to be run on other computers. Since CLEO was started fairly late in the game—mid-1961—it was able to benefit from the previous discussions on COBOL, and it is obviously influenced by work on other source languages. The aim appeared to be to start with a fairly simple language and set of facilities, and then build it up by successive issues of a revised compiler. They have, I think, been successful in this object. In September 1963 the first formal demonstration of CLEO took place (CLEO, 1963), at which a source program was compiled and the object program run; this is quite a fast rate of progress in developing the language and compiler. The first user installation was in February 1964 at the Board of Trade. At the moment there are about 25,000 instructions in the compiler. An average efficiency of about 85% is

* Lyons Electronic Office.

claimed, though this, of course, varies with the type of job done, and the measure of efficiency is a comparison with its symbolic code called "Intercode" rather than machine code. The speed of compilation is given for Leo 326 as a basic four minutes plus 100 lines of CLEO statements a minute, and the 100 lines produce an average of 1,000 machine code instructions—thus the expansion factor is 10.

Of the first 30 customers of Leo III and 326, at least 25 are using CLEO to some degree, and some, such as the Board of Trade and the General Post Office, will do all future programming in CLEO. This must be a very high percentage use of an autocode by the customers of any computer manufacturer. The proportion of CLEO programs to all programs written by users for the Leo III, 326 range is much lower—some 30% to 40%, which indicates that many users may only experiment with CLEO, rather than write all their everyday running jobs in it. The language is used in Australia and South Africa, as well as in Britain.

The number of people used to write the compiler seem quite small—from seven to ten people at various stages of its development. It is a smaller language than full COBOL or N.P.L. in that its statements are less wordy, it has fewer facilities (though it covers those in Compact COBOL), there is no report writer or sort command, only five character data names are used (since that is the size of the Leo word), and very few options are allowed, i.e. there is normally only one way of expressing a statement. To give an example of its fewer functions, there are no verbs for "Add", "Multiply" or "Subtract", instead the general expression or formula is used, e.g.

$$\text{SET X} = \frac{\text{Y} + \text{Z}}{\text{E}}.$$

The configuration required for compiling is the minimum size of Leo III, that is two divisions of store (each of 4,096 words), four magnetic-tape units, a printer and a paper tape reader.

The first three issues of the compiler have now been delivered and the fourth issue was due for April 1965. A short list of some of the things in issue four will indicate the present position of the language and show its fairly slow but steady progress. CLEO first compiles into Intercode and then to machine language. Previously these were on different systems tapes, causing a delay in operation, but now the second translation is on the back of the CLEO compiler tape so there is no operational delay in the full compiling process. Issue four permits the use of subroutines and functions with parameters; previously there were no parameter calls. More packing is permitted on magnetic tape, since

(*a*) previously all fields had to occupy a multiple of five characters even if only one character long;
(*b*) the overhead of two control words per record is greatly reduced;
(*c*) longer blocks of up to 1,000 words are permitted.

Nestings of subscripts is now permitted. Suppression of zeros can be controlled by the programmer. On the testing side, amendments to trial data or dump points may be made without recompiling the whole program.

Future issues are planned, including allowing a simple magnetic tape layout of one record to a block.

There is no scientific compiler on the Leo machines, and CLEO performs both scientific and commercial work, though mainly it has been used for commercial applications, and issue three introduced floating-point commands, though recursive functions are still not allowed.

CLEO

Example:

$$\text{SET RATE} = \frac{\text{WEEKS} + \text{EXTRA}}{\text{HOURS}}$$

Expansion: 10 : 1
Speed: 100 lines a minute
Minimum machine: 8,000 words, 4 tape units

Fig. 1

Achievement and the future

On the whole, CLEO has done well on its rather limited canvas, and its sponsors have not made the mistake of being too ambitious. At times the lack of manpower concerned with the compiler has worried potential purchasers of the machines, but perhaps a small team has proved beneficial in the end.

What the future holds for CLEO is uncertain. It may be that English Electric's new range when it appears will not use CLEO, and the machines may not be similar to the Leo machines. This trap of non-compatibility through lack of an agreed standard is very prevalent when a merger takes place. English Electric, however, have implied that they will not leave CLEO users in the lurch, but have not specified how this will be done.

TALK

English Electric, on their pre-merger series of machines, had not really launched and implemented a full-scale commercial language of their own. Their KDP10 (the RCA 501), of which ten were installed, offered R.C.A.'s COBOL, but apart from their service bureau and two other users, there was not much interest shown over a period of three to four years. Because of the R.C.A. connection, perhaps the next development may be with the SPECTRA range. For their own machines, the KDF6 and the KDF9, English Electric have introduced a language called TALK (TALK, 1964). This language is again to do both scientific and commercial applications, but is more biased to scientific work than CLEO. TALK has not yet been released to users; for the KDF6 it is undergoing field trials within English Electric, and for the KDF9 it is in its early stages of testing. The KDF9 is a large scientific computer, and the KDF6 is a medium-size general-purpose machine, so there are not likely to be many commercial users among the combined total of about 40 machines

installed or on order. Also, English Electric have implemented ALGOL for the KDF9, so most of their users who want a higher-level language will be using ALGOL.

The development of TALK started in March 1962, and by October 1963 there were fifteen people working on the two compilers. The minimum configuration required for compiling on the KDF6 is 8,000 triads of store (each of 18 bits), paper tape input and output, a monitor typewriter or line-at-a-time printer, and three magnetic-tape units. The compiling speed is quite slow at about three to four statements a minute. There is no ALGOL or other automatic-coding language for the KDF6, and it is said that it would not be possible to implement full COBOL sensibly on the machine, though Compact COBOL would be more sensible. This gives some idea of the power and size of TALK.

The minimum configuration required for compiling on the KDF9 is 12,000 words of 48 bits, paper-tape input and output, a monitor typewriter or line-at-a-time printer, and three magnetic-tape units. The compiling speed is not yet known but it is estimated that it will be about 50 statements a minute. The expansion factor is 1 to 5. As well as two versions of ALGOL, users of KDF9 can look to FORTRAN and to Mercury Autocode, so the scientific user is well catered for.

TALK does not really have full input/output editing facilities such as zero suppression or sterling format, and there is not a large measure of data description since you manipulate the data as you want it.

TALK (KDF9)

Example:

ACCEPT RATE AND HOURS AND NUMBER AND NAME
(Reads fields of data from paper tape input)

Expansion: 1 : 5

Speed: 50 lines a minute

Minimum machine: 12,000 words, 3 tape units

Fig. 2

Future prospects

With the merger it may be possible that TALK will not be generally released, and that English Electric—Leo—Marconi will concentrate on software for a new range of machines. This is because the more languages they release now, the more translators they will be committed to producing for any new machines and languages that they develop.

NEBULA

I.C.T., the largest of the British computer manufacturers, have several commercial autocoding languages as a legacy of their past, and it will be interesting to see how they settle competing claims.

"NEBULA," standing for Natural Electronic Business Users Language, is one of the most comprehensive commercial higher-level languages in the world—with far more facilities than COBOL (Braunholtz *et al.*, 1961; NEBULA). It was developed by Ferranti Ltd. for their Orion series, of which some fifteen machines have been ordered. The Orion, however, is probably not now regarded as a front-line product although users say that it has an excellent basic order code, exploited by compiler writers and keen basic programmers. Ferranti set out to produce a language for a specific machine rather than to challenge COBOL as a standard. In particular NEBULA was to satisfy the peculiar input requirements of several large companies, mainly insurance companies, to read in vast punched-card files coded in many different ways and with a wealth of over-punching built up over the passing years. Here indeed was a case of users specifying exactly what they wanted, and each new user seemed to want something slightly different, so that for some time the specification of NEBULA seemed to expand every quarter. Ferranti had wanted an open-ended language in contrast to COBOL, but I doubt if they thought at the outset that it would be as open-ended as it turned out to be before it was actually working (Rousell, 1962). The NEBULA compiler probably represents the biggest manpower investment in software in the British computer industry—some 75 man-years. Started in November 1960 it was, like most pioneer languages, late, and was not issued in its first compiling form until September 1964. About a year earlier than this the syntax-checking pass had been available. This September 1964 form omitted certain card input/output and certain obscure procedure descriptions, but did allow an amount of compiling to take place. Little testing of object programs, as opposed to compiling, occurred, however, until the fuller issue in March 1965.

NEBULA includes a sorting routine which can be used also by programmers in basic language or EMA, after setting up packed keywords for each record in accordance with certain conventions: this facility has been widely used by Orion owners since mid-1964; one user states that he can now complete in three days (with magnetic tape) sorting and tabulating routines for an extensive monthly Sales analysis which previously occupied three sorters and tabulators with summary punches for the greater part of three weeks.

NEBULA

Example:

IF OVERTIME 30 THEN PERFORM ERROR PROCEDURE
OTHERWISE GO TO SECOND STAGE

Expansion: 1 : 6 (approx.)

Speed: 3 or 4 statements a minute

Minimum machine: 8,000 words—Core, 32,000 words—Drums, 5 tape units

Fig. 3

I.C.T. say that full use of the language is now possible. The speed of compiling is slow, about three statements a minute, but I.C.T. feel this can be increased significantly in future editions. So far they have concentrated on getting the compiler working and putting in as many

facilities as possible. The minimum size of Orion for compiling is 8,000 words of core store, two drums each of 16,384 words, five magnetic-tape units, a line printer and a paper-tape punch. There is as yet little information on the efficiency of the language. There are about five users of NEBULA in the United Kingdom, two in Sweden and one in South Africa; and some compiled programs have reached over 30,000 machine code instructions. The compiler itself is very large and contains about 250,000 words—about the same as FACT (Clippinger, 1962). It was written in its own special autocode, and the compiling process did not go via the assembly language but resulted in machine code direct. Some users who had originally planned to use NEBULA re-wrote some of the programs in machine language when they found they had to wait rather a long time for the compiler.

NEBULA's future

I.C.T. have about 500 programmers on their staff—though only about 15% are working on autocodes. The new I.C.T. range—the 1900 series, which is roughly equivalent to the IBM Series 360 and probably available earlier—has now well over 150 orders, and two or three have already been delivered. This is the biggest order book that any British computer manufacturer has built up in under a year, but NEBULA is not (in early 1965) one of the languages offered on the seven machines in the 1900 range. Of course if it does prove very satisfactory in practice it could be added to the list of 1900 languages. I.C.T. will, however, have the problem of assisting NEBULA users on Orion to reprogram when and if the Orion range ceases. Seventy-five man-years cannot be lightly thrown away.

Compact COBOL

For the 1900 range I.C.T. are concentrating on Compact COBOL (Codasyl, 1964; A.S.A., 1965), that is a sub-set of full COBOL without complicated "IF" statements or formulae. This is, I believe, the first time that a British firm has standardized on this American standard. There are so many versions of COBOL at large today that it is best to specify which COBOL you mean by preceding it by the machine concerned. I.C.T.'s version of Compact COBOL is slightly larger than the Codasyl Committee's definition, in order to make it compatible with the COBOL implemented for their earlier 1301 and 1500 computers, which will be discussed later. The 1900 compiler is not written yet but is due for mid-1966.

1900 COBOL will be the first compiled into the interim phase of I.C.T.'s symbolic language called PLAN (PLAN, 1964), and then assembled by the PLAN translator. All the autocodes on the 1900 will first be compiled so that they can enter the PLAN translator. The de-bugging of 1900 COBOL is to be at source-language level, since I.C.T. claim the recompilation speed is so fast that this will be the most efficient method of testing and making alterations. The estimated time for compiling COBOL on a 1900 with a two-microsecond store of size 8,000 words of 24 bits, is 7 minutes for 500 program cards, and 35 minutes for 2,000 program cards. These times are considerably reduced by having a 16,000-word store—being 4 minutes and 11 minutes respectively—thus the size of the store makes a large difference to the compiling time. The minimum size of 1900 for compiling is roughly 8,000 words of core store, of which 5,500 words are occupied by the compiler and the rest by Executive. Also required are four magnetic-tape units, a line printer and a paper-tape reader or card reader. The expansion factor is estimated to be one COBOL statement to five machine instructions. No figures are yet available on the efficiency, but the aim is for 90%. I.C.T. plan to add to 1900 COBOL, random-access commands, and a sort command.

1900 COMPACT COBOL

Example:

```
PERFORM DISCOUNT
—
—
—
—
DISCOUNT
MULTIPLY PRICE BY 0·02 GIVING DISC.
SUBTRACT DISC. FROM TOTAL
```

Expansion: 1 : 5
Speed: 125 to 150 lines a minute
Minimum machine: 8,000 words, 4 tape units

Fig. 4

On their earlier 1300 and 1301 series, of which some 200 have been sold, I.C.T. offered "RAPIDWRITE" which was really a minimal version of COBOL, using pre-printed and pre-punched cards rather than coding sheets to help cut down the verbosity of COBOL (Humby, 1962). The first version did not turn out to be very efficient, and you need a large amount of desk space to see all the program cards at once and hence the flow of your program, especially when making alterations. Later versions are said to be more efficient. There are only about five customers using RAPIDWRITE on the 1301, but the 1900 Compact COBOL compiler can accept RAPIDWRITE input, and I.C.T. think it is a worthwhile project.

Compatibility among I.C.T. machines

Required COBOL 61 (COBOL 61) plus some options is working on I.C.T.'s 1301 and 1500, though the options are not always the same on both machines. The 1301 compiler was developed by I.C.T., and the 1500 compiler by R.C.A., since this is the British name for the RCA 301—hence the reason for certain differences in source language and compiler techniques on the two machines. There are some thirty 1500's (out of the

total of about 130 sold) in South Africa, Central Africa and Australia, and about 20 of these installations use 1500 COBOL because of the shortage of programmers in those areas. The language is said to work very well but the compilation may take two hours or more. I.C.T.'s experience of 1301 COBOL and 1500 COBOL has lead to the development of 1900 COBOL with its emphasis on making the language as simple as possible, cutting out a lot of options and permitting compatibility at the highest possible common level, though this level may be fairly low. But I.C.T. have now announced a second phase including facilities more akin to 1500 COBOL in a fuller version of 1900 COBOL. I.C.T., like American firms, are now facing the problem of compatibility (Share Committee, 1958). Though there have been relatively few autocode users among their 330 sales of the 1500, 1300 and 1301, they wish to make it possible for autocode users to change to the 1900, especially as COBOL users tend to be the more forward-looking firms who are more likely to make an early change to the 1900 series. I.C.T. state that it will be possible to write and compile a program in Compact COBOL or RAPIDWRITE on the 1301, and then have a fully tested program ready for compilation by the 1900 Compact COBOL compiler.

The 1900 series will also have FORTRAN IV, and a paper-tape version of FORTRAN II is already working, though the magnetic-tape version is still being written. Extended Mercury Autocode (EMA) will also be available—this is a scientific language quite common in the United Kingdom (Brooker, 1958)—and work will be started on an ALGOL compiler.

Atlas commercial language

I.C.T. also have a very large computer developed by Manchester University with Ferranti Ltd. called Atlas (Kilburn *et al.*, 1961; Howarth *et al.*, 1961). At present, they do not seem to be offering any commercial autocode on this—neither COBOL nor NEBULA. But one user of Atlas—the Institute of Computer Science (London University)—is developing the Atlas Commercial Language—ACL. This again seems to owe a lot to COBOL and ALGOL, though the intention is to make a functional language rather than a procedural language. For instance, its authors wish to say "Print (A + B − C)" in one command, rather than doing the arithmetic work in one part of the program and printing the results in another part. This feature is not in the compiler at present, since only a sub-set of the ultimate intentions have been developed so far.* The compiler is working, but up to a short time ago no complete suite of programs had been fully developed. The compiler took only three people a year to write—which must be one of the smallest investments in autocodes anywhere in the world. The "Compiler Compiling Language" of Brooker and Morris (Brooker *et al.*, 1963) was used to write ACL, and the source language is compiled direct into machine language without going through any intermediate phase of an assembly language. De-bugging is done in the source language.

At the moment ACL seems rather limited for large commercial jobs, since it has been implemented for only paper-tape input not punched card input. There are no reserved words but key words are underlined—this is all right for paper-tape input but makes it very difficult to punch the language into cards. This seems to indicate poor research on behalf of the authors as to what commercial firms actually use for their program input media, and tends to throw doubts on whether their other facilities are any closer to the needs of the business programmer.†

BABEL

There are, of course, other commercial languages produced in Britain which have not been discussed in this paper (Cormack, 1962; Cormack, 1965; Barron, 1963). ACL also seems to add one more language to the tower of Babel of commercial automatic-programming languages in the United Kingdom. This in many ways seems an unsatisfactory situation for users and manufacturers alike. No customer wants the expense of rewriting all his programs in a new language when he invests in a more modern machine. No manufacturer wants the expense of developing compilers for all his rivals' languages (and his own) to attract their old customers.

Operating systems

Operating systems were also out of favour with British manufacturers for a long time, it being thought that it was sufficient to allow each programmer to operate his program in the way he wanted. This not only pleased individual programmers developing their routines but saved manufacturers from having to invest large sums in the writing of the software to give a standard operating system. But the advent of time-sharing machines and many different languages has led British manufacturers to introduce operating systems (Howarth, 1962; Cloot, 1965; Bouvard, 1964). On the 1900 series the full operating-system software will only be offered on the larger, faster machines—that is the 1906/7 (1·25 or 2·25 microsecond store). The size of the program is not yet known since it has not been finished, but it is expected to be about 20,000 to 30,000 words, including the use of 9,000 words of core storage, and the rest on drums or discs. I.C.T. feel that a core size of 32K words will be required to take advantage of a full operating system. The features that are planned for the 1906/7 operating system include:

(*a*) Store a list of requests for jobs.
(*b*) Amend the list of jobs, including modifying parameters.

* The *Print* (*Arithmetic Expression Result*) feature is available also in Extended Mercury Autocode.

† Nebula and EMA programs for Orion and EMA programs for Atlas are being punched into 5-track or 7-track tape by all users currently.

(*c*) Initiate jobs when storage and peripherals become available.
(*d*) Adjust priorities to optimize overall throughout.
(*e*) Store temporarily input and output data for a job on a file-storage medium selected by the operating system, or on a device allocated direct by the programmer.
(*f*) Suspend jobs if they exceed allocation of running time or output volume.
(*g*) Time automatically each program for charging purposes.
(*h*) Handle several source languages and call in the correct compiler from the library.
(*i*) Allow multiple on-line consoles.

Similar systems have already been written and are working for I.C.T. Orion (OMP, 1964) and Atlas (Howarth, 1962) and the lessons learnt on them (sometimes painfully) are being used for the 1900 operating system (Heatherington, 1965).

The Orion Monitor Program is now very successful and is proving how useful such a system is.* Leo III and 326 also have an operating system which has many similarities to the 1900 series, and this is already running successfully (Lewis, 1965).

Conclusion

In conclusion it can be said that the United Kingdom users and manufacturers have reached a very similar situation to that in the U.S.A.—operating systems and autocodes are now available, and problems of compatibility or moving from one machine to the next are becoming more important. Probably less than 5% of British users, however, write in commercial autocoding languages. Because of the history of the computer market there are many commercial English languages available, and it could be in the interests of the users and of the two largest manufacturers (I.C.T. and English Electric—Leo—Marconi) if a standard could be found, thus limiting everyone's investment. Since there is little sign that the standard will be COBOL, perhaps it will be N.P.L. or to give it its new name PL/I (PL/I, 1965).

* It occupies 512 words core and some 7,500 words drum on most Orions.

References and Bibliography

A.S.A. (1965). *X*3.4 *COBOL Information Bulletin No.* 6, *COBOL Standards*, American Standards Association, May, 1965.

Barron, D. W., *et al.* (1963). "The Main Features of CPL," *The Computer Journal*, Vol. 6, p. 134.

Bouvard, J. (1964). "Operating System for the 800/1800," *Datamation*, May, 1964, p. 29.

Braunholtz, T. G. H., *et al.* (1961). "NEBULA—a Programming Language for Data Processing," *The Computer Journal*, Vol. 4, p. 197.

Brooker, R. A. (1958). "The Autocode Programs developed for the Manchester University Computers," *The Computer Journal*, Vol. 1, p. 15.

Brooker, R. A., MacCallum, I. R., Morris, D., and Rohl, J. S. (1963). "The Compiler Compiler," *Annual Review in Automatic Programming*, Vol. 3, p. 229. Published by Pergamon Press.

Clarke, B., and Felton, G. E. (1959). "The Pegasus Autocode," *The Computer Journal*, Vol. 1, p. 192.

Cleo (1961). *CLEO*, Leo Computers Ltd., November, 1961.

Cleo (1963). "Practical Implications and Current Development of the CLEO System," *Demonstration* for Leo Users Association, Leo Computers Ltd., September, 1963.

Clippinger, R. F. (1962). "FACT," *The Computer Journal*, Vol. 5, p. 112.

Cloot, P. L. (1965). "What is the use of Operating Systems?" *The Computer Journal*, Vol. 7, p. 249.

Cobol—60. *Report* published by Department of Defense, Washington D.C. (April 1960).

Cobol—61. *Report* published by Department of Defense, Washington D.C., 1961.

Codasyl (1964). "COBOL—Preliminary Edition (1964)," *Conference on Data Systems Languages.*

Conference Discussion (1962). "Conference on Automatic Programming Languages for Business and Science," *The Computer Journal*, Vol. 5, p. 121 and p. 170.

Cormack, A. S. (1962). "Early Operating Experience with Language H," *The Computer Journal*, Vol. 5, p. 158.

Cormack, A. S. (1965). "Inter-action between user's needs and language—compiler—computer systems," *The Computer Journal*, Vol. 8, p. 8.

Cowan, R. A. (1964). "Is COBOL getting cheaper?" *Datamation*, June, 1964, p. 46.

D'Agapeyeff, A. (1962). "Current Developments in Commercial Automatic Programming," *The Computer Journal*, Vol. 5, p. 107.

D'Agapeyeff, A. (1965). "Software in Europe," *Datamation*, May, 1965, p. 31.

Ellis, P. V. (1961). "COBOL", *The Computer Bulletin*, Vol. 4, p. 144.

Gearing, H. W. (1961). "The use of Pegasus Autocode in Some Experimental Business Applications of Computers," *The Computer Journal*, Vol. 4, p. 30.

Gill, S. (1959). "Current Theory and Practice of Automatic Programming," *The Computer Journal*, Vol. 2, p. 110.

Heatherington, G. A. (1965). "What is an Executive Program?" *The Computer Bulletin*, Vol. 8, p. 139.

Howarth, D. E., *et al.* (1961). "The Manchester University Atlas Operating System, Part II User's Description," *The Computer Journal*, Vol. 4, p. 226.

Howarth, D. E., *et al.* (1962). "The Atlas Scheduling System," *The Computer Journal*, Vol. 5, p. 238.

Humby, E. (1962). "Rapidwrite—a New Approach to COBOL Readability," *The Computer Journal*, Vol. 4, p. 301.

KILBURN, T., *et al.* (1961). "The Manchester University Atlas Operating System, Part I Internal Organisation," *The Computer Journal*, Vol. 4, p. 222.
LEWIS, J. W. (1965). "The Management of a Large Commercial Computer Bureau," *The Computer Journal*, Vol. 7, p. 255.
MILLER, J. T., *et al.* (1965). "Comment," *Computer Survey*, Vol. 4, p. 123.
NEBULA. "A programming language for Commerical Data Processing," *Manual* published by I.C.T. Ltd.
OMP (1964). "ORION Monitor Program," *Data Processing*, November–December, 1964, p. 318.
PAINE, R. M. (1960). "Automatic Coding for Business Applications," *The Computer Journal*, Vol. 3, p. 144.
PAINE, R. M. (1962). "Waiting for COBOL," *Automatic Data Processing*, Vol. 4, p. 21.
PL/I (1965). *I.B.M. Operating System/360. PL/I Language Specification*, I.B.M. Systems Reference Library, File No. 5360–29. Form C28–6571–O, 1965.
PLAN (1964). *PLAN Training Manual*—1900 *Series*, I.C.T. Ltd., Technical Publication 3144, 1964.
ROUSELL, A. R. (1962). "A Progress Report on NEBULA," *The Computer Journal*, Vol. 5, p. 162.
SHARE AD HOC COMMITTEE ON UNIVERSAL LANGUAGES (1958). "The Problems of Programming Communication with Changing Machines," *Communications of the A.C.M.*, Vol. 1, p. 12, and Vol. 1, p. 9.
SOAP II (1957). "*SOAP II*," *Programmers Reference Manual*, I.B.M. Publication 32–7842, 1957.
STEEL, T. B. (1960). "UNCOL, Universal Computer-Orientated Language Revisited," *Datamation*, Vol. 6, p. 18.
TALK (1964). *TALK*—6 *Programmer's Reference Manual*, English–Electric–Leo Computers Ltd., February 1964.
THOMPSON, T. R. (1962). "Fundamental Principles of Expressing a Procedure for a Computer Application," *The Computer Journal*, Vol. 5, p. 164.
WILLEY, E. L., *et al.* (1961a). "A Critical Appraisal of COBOL," *The Computer Bulletin*, Vol. 4, p. 141.
WILLEY, E. L., *et al.* (1961b). "Some Commercial Autocodes—a comparative study," *APIC Studies in Data Processing No.* 1. Academic Press.

Correspondence

An impossible program

To the Editor,
The Computer Journal.

Sir,

It seems to me that a point has been missed by your correspondents on this subject. I too, think that Mr. Strachey's proof* is at fault, for this reason:

The Program P includes the procedure $T(P)$ and it is possible for some particular argument of T that T may itself loop, so that since P includes $T(P)$ the investigation $T(P)$ must include the investigation $T[T(P)]$, and this must include an investigation of all parts of $T(P)$ including $T[T(P)]$ i.e. $T[T(P)]$ includes $T\{T[T(P)]\}$, which must in turn include an investigation of all parts of $T[T(P)]$ including $T\{T[T(P)]\}$ and so on.

When P is executed an infinite recursion with no means of escape will result, i.e. a closed loop exists in T when its argument is P.

P will loop, but since the loop is internal to T, $T(P)$ will not take any value and there is no contradiction. It is clear that if T exists it is restricted to investigating programs not including itself, but no proof of the impossibility of *this* program has been given.

Yours faithfully,

B. E. BOUTEL

34 The Hoe,
Carpenders Park,
Watford, Herts.
10 August 1965.

* This *Journal*, January 1965, p. 313

To the Editor,
The Computer Journal.

Sir,

I must apologize to Mr. ApSimon. I did not intend* to ask him to accept a non-existent proof involving a hypothetical fraction. I actually intended to refer to a non-existence proof of a hypothetical function, but my handwriting seems to have betrayed me.

Yours etc.,

C. STRACHEY.

Churchill College,
Cambridge,
5 August 1965.

To the Editor,
The Computer Journal.

Sir,

Mr. Strachey seems to have proved that, if R may include $T(R)$ and the function $T(R)$ always terminates, then the function $T(R)$ does not exist. Are these useful conditions? May not $T(R)$ without one or both of them exist?

Yours faithfully,

J. H. G. PHILLIPS.
MICHAEL IRISH.

The National Cash Register Co. Ltd.,
88/92 Baker St.,
London, W.1.
23 September 1965.

* This *Journal*, July 1965, p. 176.