Handling of Sterling

Being of American origin COBOL, of course, makes no provision for the handling of Sterling quantities. Moreover, the addition of such facilities to the scheme cannot efficiently be carried out using the COBOL language. At English Electric we have therefore provided subroutines written in an assembly language (own code) for conversion to and from Sterling.

Conclusions

COBOL has been found to be fairly easy to learn, but the old rule of "the better the programmer the better the program" still applies.

COBOL produces first-class program documentation. This is very important and normally quite difficult to obtain from programmers.

Whatever the limitations of COBOL (and it has a few), and however much people decry it, the following facts stand out.

It exists.

It is better for commercial programming than most machine codes.

It is easy to use.

It has been implemented.

It is a big step towards a real Common Business Language.

In a bureau where many different programs are run daily the standardization of operating and re-run instructions has proved to be extremely useful.

Acknowledgement

This paper is published by kind permission of The Manager, Data Processing and Control Systems Division, The English Electric Co. Ltd., Kidsgrove.

Early operating experience with Language H

By A. S. Cormack

This paper assumes a basic knowledge of the fundamental principles of automatic programming languages, and is intended to show some of the practical advantages that have been obtained by using such a language. A brief description of Language H and the factors which influenced its development is included, but the main emphasis lies on the actual operation of the system as a whole.

Introduction

With the current emphasis on COBOL as a Common Universal Business Language, and the considerable effort that is being made to get COBOL accepted by all manufacturers, it is not surprising that a certain amount of criticism is levelled at those manufacturers who have taken the decision to design and implement their own languages.

In answer to this criticism most manufacturers have agreed to offer COBOL as well as their own language. The choice is then up to the user to decide which one is the more suitable for his particular purpose.

The major factor which influenced the decision to proceed with the development of Language H stemmed from a consideration for the smaller machine user. Most of the effort so far has been directed towards comprehensive languages for large installations, and although it is obviously possible to write compilers for this type of language for a small machine, the number of runs and the time needed for translation do not make this a practical proposition.

The suggested method of taking a subset of the complete language is, at best, only a partial solution. The difficulty of extracting an effective subset without in some way destroying the logical completeness of the language is almost as great as designing an entirely new language.

Basic philosophy

The main aim behind the development of Language H was one of simplicity. It was hoped that the version currently running would prove to be the minimum effective system, and to achieve this a greater part of the work was directed towards deciding, not what to include, but what to exclude. This version, although complete in itself, is intended to be a foundation upon which the language will be allowed to grow naturally. It is envisaged that expansion will depend as much upon field trials and suggestions from users as upon the compiler writers themselves. In this way it is hoped to achieve eventually a multi-level language which can be truncated at natural logical stages to suit the particular requirements of different installations. without in any way affecting the structure of the language.

Brief description

Language H is a simple, single level, procedural language allowing one level of subscripting, in which many data-processing problems may be expressed. No separate description of the data being handled is required of the user—sufficient information is obtained by implication, from the way phrases are written, to provide a range of checks on validity and to enable appropriate machine-code to be produced.

The conventions required to meet the needs of filing and input and output specifications are simple and straightforward.

Some of the requirements borne in mind in developing this approach have been the following:

(1) The smallest possible number of effective phrases should be provided.
(2) The load on the memory of the user should be kept as small as possible.
(3) Any program should be unambiguous.
(4) The structure of the language should be independent of any particular computer.
(5) The unit of information handled should be of the size of a number, a name, or a short comment; only exceptionally is it, on the one hand, a single character or, on the other, a record.

In spite of the simplicity of the language it is not as naïve as perhaps it may appear from the description. It has most of the facilities which people have been led to expect from autocoding languages, such as:

(*a*) nested repeat loops;
(*b*) conditional branch commands;
(*c*) provision for segmentation;
(*d*) a perform command to permit sections of program to be obeyed out of sequence;
(*e*) one level of subscripting for list work.

Some of the proposals for the next version include provisions for sorting, random access, the increase of levels for subscripting, and the extension of the file commands.

The example in Fig. 1 is an extract from the Language H Payroll program mentioned later in the paper. It is not a complete section and is, by itself, meaningless, but it is intended to illustrate the way in which the language is used to handle data-processing problems, and to give an indication of the effect of some of the commands.

It will be noticed that punctuation is used merely to increase readability and does not have the syntactic meaning which some other languages assign to it. In other words, it is not necessary to remember that, say, a full stop terminates an imperative sentence and that a comma terminates a conditional expression or that an imperative statement is terminated by a semi-colon. The syntax of Language H is defined mainly by the way in which the commands are written.

Operation of the compiler

The compiler, which contains some 20,000 instructions, has been designed to run with minimum operator intervention. After loading the compiler on one of the magnetic film units, a short entry tape, which is read in under the initial instructions, initiates the compiling run, and thereafter the operation is entirely automatic.

During translation extensive checks are carried out to test the validity of the source program, and comments are printed to indicate the nature of the error found, if any. These comments are prefaced by the number of the command in which the error has been found and the chapter in which it has occurred. At the end of the translation additional checks are carried out to test that every jump command has a destination and vice versa, and additional information is printed indicating the number of chapters, operands and flags used by the source program.

The second part of the compiler, which generates the object program is then read down and compiling continues.

During this section of the operation all the labels are printed out with their associated addresses, to act as a guide during the testing of the object program. At the end, further comments are printed indicating the start and end of the program, which film it is held on, and which key to press for a post-mortem of the object program, output of the object program for use on another machine, or optional load and go.

If the program is required to run on another machine a self-triggering binary tape is produced headed by a short "boot-strap" program which enables it to be read in under the initial instructions of the object machine.

Various diagnostic routines are provided for examining the operand, literal or destination lists, and it is intended to produce an abbreviated version of the source program during translation to shorten the time of any recompilation that may be necessary.

Correction of the source program

At the moment any faults that are detected during the translation run due to punching or syntatic errors may be corrected either by recopying the tape, or by overpunching a control symbol at the place where the fault occurs and repunching a short piece of tape carrying the correct information.

It is debatable whether there is any advantage to be gained by increasing the key-word list by the addition of compiler directives to enable corrections to be read in before compiling begins and inserted during translation, but provision has been made for this should it be found to be necessary.

Testing the object program

On the assumption that the compiler is correct and that no source-program errors have escaped detection during compilation, the object tape will be a valid machine program. If it does not produce the expected results this will be due either to logical errors on the part of the programmer, or to an incomplete understanding of the effect of the commands described in the manual, or to an inadequacy of the manual itself.

There are two methods which have been found in practice to be the most helpful in debugging object programs. The first is simply to use the label addresses printed out during generation of the program to set up selected stops and thus trace the course of the program step by step from start to finish. The second is used where a lot of internal calculation is involved, and

Entry.	Get block no. A, type A, ref. no. A (and up to 57 cells) from channel 1. Add 4 to type A. Branch to entry, output film error, block error, during payroll, before payroll, leaver, summary or output film error according to type A.
Block error.	Print "100."
Type minus four.	Subtract 4 from type A.
Type.	Load something with type A.
Out.	Print number (max. 999) being something and go to stop.
Output film error.	Print "101," and go to type minus four.
During payroll.	Get block no. B, type B, ref. no. B (and up to 15 cells), and code (and up to 41 cells) from channel 1. Turn on A flag if ref. no. A is equal to ref. no. B. Go to test leaver if A flag is on. Print "102."
Error ref. no.	Print number (max. 999999) being ref. no. A and load reference with ref. no. B.
Error ref.	Print number (max. 999999) being reference.
Back entry.	Wait. Go to entry if switch 24 is off. Go to stop.
Test leaver.	Go to leaver adj. if leaver flag is on. Go to salary if output flag is on. Go to ref. if input flag is on. Go. to input if ref. flag is on.
NCR Graduated.	Turn on A flag.
NCR	Load NCR pension with taxable pay for-month and subtract £21 : 0 : 0 from NCR pension. Go to nothing if NCR pension is negative. Divide NCR pension by 20 and go to normal if A flag is off.
Graduated.	Load grad. pension with taxable pay for-month and subtract £30 : 0 : 5 from grad. pension. Go to none if grad. pension is negative. Calculate grad. pension (function 51) from grad. pension. Turn on A flag if grad. pension is greater than £1 : 2 : 1. Go to normal if A flag is off. Load grad. pension with £1 : 2 : 1 and go to normal.
Released refund.	Subtract suppressed tax refund from tax for-month.
Pensions.	Go to nothing if superannuation code is zero. Turn off A flag and branch to NCR, graduated or NCR graduated according to super-annuation code. Print "111." Perform error ref. no. Perform error name. Load something with superannuation code and perform out: wait. Go to stop if switch 24 is on.
Nothing.	Load NCR pension with 0.
None.	Load grad. pension with 0.
Normal.	Go to usual if normal stamps are negative.
NHI Code.	Turn on A flag if nhi stampvalue code is equal to numeral for pay. Go to nhi if A flag is on. Repeat 21 times from nhi code counting in pay. Print "112," perform error ref. no. and perform error name. Load something with nhi stamp value code and perform out: wait. Go to stop if switch 24 is on.
Total ded.	Calculate total ded. (function 1) from hsa ded., sports club ded., endow. ded., special ded., tax for-month, NCR pension, grad. pension, and NCR vol. contrib. Add national insurance to total ded. Calculate gross (function 1) from overtime, bonus, and salary for-month; calculate net salary (function 3) from gross, expenses, total ded., and float repayment.

Fig. 1.—Part of a payroll program written in Language H

consists of inserting commands to print out intermediate results at various stages. These commands can be deleted and the program recompiled when testing has been completed.

Operating experience

The first compiler was written for the National-Elliott 405M with the primary object of testing the soundness of the language, and was completed by August 1961. It was checked out and found to be effective by December 1961. During this time one or two small production jobs had been run, together with a rather more sophisticated program to play noughts and crosses. This had no particular value as a commercial program but proved to be useful in checking the multiple branch, repeat, and perform commands in conjunction with the subscripting facility.

Work was begun in January 1962 on a compiler to run on the National-Elliott 405M and produce object programs to run on the National-Elliott 803B. At the

same time it was decided to run a full-scale production job using Language H at the beginning of May 1962. For this purpose a monthly pay-roll for 550 people, divided into two categories, was chosen and the specification was received at the end of January.

The major portion of the program was written by a person with no previous machine programming experience, and the first drafts were ready by the middle of March 1962. Checking, testing and revision of the program occupied a few days, and the program was completed and compiled by the end of the month. Two versions of the source program were written, one to run on a National-Elliott 803B with 4,096 words of core store and three films, and one to run on the same type of machine with 8,192 words of core store.

Very little trouble occurred during the testing of the object programs, and they were all working by the third week in April. This time would have been considerably shortened had it not been for the delay caused by the checking out of the compiler.

Table 1 was drawn up during the preparation of the payroll to give an indication of the time taken to compile source programs in Language H.

Table 1

Language H payroll program: source program timings

Permanent amendment run:	
Number of commands:	1,000
Time taken to compile (in minutes):	27
Number of machine code instructions produced:	9,000
Calculation and payslip output run:	
Number of commands:	560
Time taken to compile (in minutes):	19
Number of machine code instructions produced:	7,000
Credit transfers, bank lists, leavers records, summaries and tax refund output run:	
Number of commands:	365
Time taken to compile (in minutes):	17
Number of machine code instructions produced:	5,000

Efficiency

Comparisons drawn from trivial problems programmed in both source language and machine code are clearly of little value, and the time and effort necessary to prepare a wide variety of complex problems in both forms precludes this method for estimating the efficiency of object programs generated by Language H. However, work is in progress at the moment on the preparation of a source program in Language H for a production job currently running on the National-Elliott 803B. It is hoped, therefore, to have some figures on the relative efficiency of both these programs by the end of May 1962.

It is perhaps worth mentioning at this stage two of the techniques that have been used in an effort to increase the efficiency of the object programs. No claim is laid to originality in these techniques, and they are included merely to indicate the lines on which work is proceeding in this field.

In the first place, care has been taken to ensure that the generation of the object program is purely dynamic, that is, at no stage is a macro command included that has not been specifically called for by the source program. In other words, as far as is possible there is no superfluous information in the object machine at run-time. This is clearly of primary importance when dealing with smaller machines.

The second point is directed towards achieving a balance between space and speed. Clearly, the way that this is done depends to a great extent on the type of machine that is being considered. In the case of the National-Elliott 803B, which has a comparatively large core store, it was decided that more emphasis should be placed on speed than space. Since, however, optimum timing tends to produce lengthy macro generation, a facility has been included to ensure that any of the longer macros are only put in once however many times they may be called for subsequently. This facility which is, in fact, automatic subroutining, has been extended across any number of levels so that macros may use parts of other macros which may, in turn, use parts of other macros and so on. In this way it is possible to ensure that there is a minimum of repetition in the object program.

Conclusions

Practical results have indicated that Language H is a useful tool for solving a wide number of data-processing problems. In common with other autocoding languages it is only a tool and as such must be handled with care and precision. Any assumption that it can be used carelessly to solve any problem (as a sort of problem-solving panacea) should be strongly discouraged, as this approach can only lead to disaster.

Finally, one interesting feature emerged from operating experience with Language H. That is that people with no previous experience of machine-code programming appear to be more readily able to grasp the essentials of autocoding languages than experienced programmers. This may be due to the programmer's reluctance to wade through a manual step by step until he has thoroughly mastered it, or possibly to misunderstandings caused by assumptions made in the light of previous programming experience.

Bibliography

"COBOL 60." US Department of Defence (April 1960).