Conference Proceedings

On the following 36 pages are published the Proceedings of Sessions 2 and 4 of the Conference on **Automatic programmin languages for business and science**, held at the Northampton College of Advanced Technology, London, on 17–18 April 196. Proceedings of Sessions 1 and 3 were published on pp. 107–139 of the July 1962 issue.

Operating experience with COBOL in a service bureau

By M. A. Kingsbury

Introduction

COBOL has been in use at the Central Computing Service, English Electric Co. Ltd., Kidsgrove, on a KDP10 computer since October 1961. This has given us experience in writing and testing programs in COBOL, and in running the object programs. It has enabled us to find out how long programs take to compile, and how efficient the object programs are. We have also been able to decide how long a course is needed in order to teach COBOL.

Courses

COBOL courses at English Electric have been of either three days duration or two weeks. It has been found that people can learn—and, in fact, have had to learn COBOL in three days. However, it is generally felt that a two-week course is better; the first week is spent in learning COBOL, and the second in writing and testing a simple program under proper supervision.

Writing and testing COBOL programs

Defining a problem and producing functional flow-charts is not accelerated by COBOL; however, once these have been done the time taken to write a program is greatly reduced, by, I would guess, a factor of 4 to 1. It has been found that, after a little encouragement, programmers write the two-dimensional functional flow-charts in simple COBOL statements. This means that they then have a one-to-one correspondence between the functional flow charts and the procedure division of the program.

One finds that, in doing this, the complete logic of the program is checked at the functional flow-chart stage, and writing of the procedure division consists simply of copying the flow chart.

Basically the compiler is split into two sections. The first section checks the COBOL program for all programming errors such as trying to move alphabetical information into a numeric field, moving large fields into small ones, incorrect program format, use of non-existent data names, etc.—it also produces an updated edited copy of the COBOL program which includes all corrections. The second section produces the actual machine coded object program.

On testing a reasonably sized program of 2–500 COBOL statements one should get a compiled program after two or three runs on the computer. This is not because people don't make mistakes when programming in COBOL, but because the Compiler prints out most of the COBOL programming errors—it doesn't stop at the first error but lists them all. Some of the most common ones detected during compilation are:

Data names not defined in the data division.
Sizes of data fields incompatible.
Data names mis-spelt.

When a program has been compiled a complete description of it is printed out. This includes the locations in the high-speed memory of all work areas and file areas, and a list of all the machine instructions produced for each COBOL statement. This complete program-description is chiefly used when testing the program, or when changing or correcting it some time after it has been written.

We have found that testing object programs is fairly simple. This is because the only errors which should exist in programs are systems errors. Thus one finds that object programs nearly always produce output—not necessary correct output, but, nevertheless, output which can be analysed.

The accepted way of testing object programs has been to run them on test data and to examine the results. If this is not sufficient to determine the cause of error one can stop the object program at any designated point and take a print-out from the high-speed memory. Using this, together with the Compiler listings, one can examine the contents of all input and output areas, work areas, etc. In order to correct an object program, and to retain it as a COBOL-produced program, all corrections must be applied by recompilation. However, it has been found worthwhile (i.e. it saves computer time) to make corrections to the source program in COBOL and to the object program in machine code. When the object program works the source program is recompiled in order to produce a correct object program.

One of the side benefits of COBOL is that it provides most of the housekeeping required for commercial programming; this covers tape interchanging, tape labelling re-run procedures, etc.

Efficiency

The best machine coded program should always be more efficient than the best COBOL program.

In general, on file processing jobs, COBOL produced programs compare very favourably with m/c coded programs. The difference between COBOL and m/c coded programs on other problems really depends on the nature of the problem.

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.