In these courses we cover the whole of ALGOL 60, including a thorough discussion of "Jensen's Device" and of recursiveness (or, more generally, nested activations of the same procedure). I mention this, because it is sometimes pointed out that these "advanced features of ALGOL 60" will frighten and repel potential users. Our experience quite definitely points in the opposite direction: the audience was thrilled by them every time the course was given. In practical computations these features are not too frequently used, but the bare fact that the programmers could use them if they wanted to made the language very appealing.

We have given the course four times, with a total number of about 240 participants, and one may ask how many machine users have been created in this way. For our own installation it is about thirty people, fifteen of whom are to be considered as regular machine users; the other fifteen turn up at less frequent intervals. I consider this a very good result, because these thirty people had to be recruited from those attendants that did not have their own machine at their disposal. At least one half of the participants were from other computing centres.

Actual use of our ALGOL 60 system is steadily increasing. I cannot give figures for other X1 installations, which have received copies of our translator. At the installation at the Mathematical Centre we started with 20% machine time spent on ALGOL programs; in the meantime this has been increased to 50%.

Omission of syntactical checking in translation has proved to have been a grave error. Every user finds that his first program contains a number of silly, clerical errors. This number of errors per program decreases very fast as the programmer gets more experience, and it is therefore my impression that it is hardly worth the trouble to let the translator look for the next error after the first one has been found. The omission of syntactical checking is the more regretful as it could have been incorporated at so little expense.

Furthermore, we find that the program for a particular problem is often processed in a couple of successive versions. Roughly: the first version is just plainly wrong, because it contains some logical errors, neglect of some exceptional cases, etc. The second version works, but the programmer is not satisfied with its performance. In the third version the programmer, who in the meantime understands his problem better, improves his strategy, and in a fourth version he improves on the programming. This is more or less the background of the fact that our "irregular users" suddenly turn up four times within a period of, say, two weeks; then we don't see them for quite a long time, but usually they return sooner or later . . . with their next problem. This experience is very encouraging.

The run-time system has no additional diagnostic facilities. We could include them, but from the fact that we have not done so one can deduce that the need for them is regarded as insufficiently urgent. In the case of longer programs it is quite usual to insert some conditional output statements in the earlier versions of the program. If they are enclosed between two pieces of blank tape on the input tape it is a trivial operation to remove them in the final version.

Furthermore, there is no possibility of a "post-mortem dump." There is no point in just printing out the contents of the store: as storage allocation is fully dynamic these data would be too hard to interpret. If a post-mortem dump were to be of any value it would have to produce the values of variables in store together with their identifiers in source language. This, however, would imply the availability of the complete "identifier table," but this is nowhere available in its entirety, not even during translation (this is only account of storage limitations).

The translator gives no print-out of the object program, again because there is no point in it. The structure of the object program has so little in common with that of handwritten programs that with a thorough knowledge of just ALGOL 60 on the one hand and just the X1 on the other, the print-out of the object program still won't be very helpful. As a consequence, all modifications and corrections must be made in the source-language program: we have made it virtually impossible to correct or to modify the object program, and we have done so on purpose. Some people like to have this possibility in order to avoid retranslation, but we regard this as an obsolete technique, which is not to be encouraged. Quite the contrary.

Report on the Elliott ALGOL translator

By C. A. R. Hoare

The Elliott ALGOL programming and operating system has been designed to suit the needs of an Elliott 503 computer installation which allocates at least part of its time to running programs on a service basis. The main problem in operating a computing service is to maintain a high average number of programs processed in a given period, and in particular to reduce to a minimum the time spent in changing over from one program to the next.

A number of published methods of tackling this problem have involved the use of buffers, interrupt lines, wired-in programs, time-sharing, and other sophisticated hardware and software devices. The aim of the Elliott system is to provide an acceptable operating method for an installation which is interested in the high speeds of second-generation computers, but does not wish to be involved in the heavy capital outlay and maintenance costs of high-volume backing stores and other peripheral

devices. It is this which provides an explanation for the differences between the Elliott system and certain other systems at present projected or in existence.

In the first place, we could not accept the inefficiency of an interpretive system, which has been favoured by many compiler writers whose needs differ from ours. On a computer which has no built-in facilities for floating-point arithmetic, it has been found that the use of an interpretive scheme will reduce the efficiency of a program by a factor of up to six. On the 503, which performs floating-point arithmetic by hardware, the use of interpretation would reduce efficiency by a factor of about thirty. This is too high a price to pay for the other advantages of interpretation.

We have also rejected the use of a multiple-pass translating system, which was used by the designers of FORTRAN and other autocodes. The time taken for the output and re-input of the results of intermediate passes would in many cases be longer than the time taken to run the program, especially where paper tape is the only available output medium. It is expected that many of the problems presented to a service installation will be no larger than those which have been solved on slower machines in the past. Many such problems will be solved in under a minute on the 503. The advantages of this great increase of speed will be lost if the program takes a long time to read in and translate.

System Design

The most suitable translating system for our purposes is one which accepts a source program in ALGOL, reads and translates it at the full speed of the paper-tape reader (1,000 characters per second), and immediately transfers control to the translated program. The normal practice will be to leave the translator permanently in the working store, and to read in and execute programs one after the other with minimum delay. When a program requires a large amount of working space, it will overwrite the translator, and then the translator will have to be read into the store again before the next ALGOL program can be read in. A two-pass system will be used to translate only those programs which are so long that they will not fit into the store of the computer together with the translator.

The over-all productivity of an installation which uses this operating method will depend critically on the frequency with which the translator has to be re-input, and the frequency with which the second pass becomes necessary. This frequency depends largely on the length of the translator. The necessity of keeping the length of the translator within strict bounds was the main factor in the adoption of two important decisions, which might otherwise appear to detract from the usefulness of the Elliott system. The first of these decisions was to place some restrictions on the full generality of ALGOL 60, and the second, to avoid optimization of all but the most rudimentary kind. The second decision was easier to take, since the high internal speeds of the 503 will ensure that most programs will be input and output limited, especially in the case of a minimum (or near minimum) configuration computer.

The restrictions which we have considered necessary affect facilities which are unlikely to be needed or even understood by non-professional programmers, who will provide the greater part of the work of a service installation. When the need arises to adapt a published program, this may be done fairly easily by a competent programmer. Full details of all restrictions, together with instructions on the easiest methods of getting round them, will be distributed to all users of the system.

Construction of Translator

The method adopted in constructing the translator was also conditioned by the extreme need for brevity. The first sections to be written were those which read in the consecutive identifiers, numbers, and basic symbols of an ALGOL program, and those which administer a simple relative-addressing scheme for the output produced by the translator. Then we constructed a routine which processes the declarations, and produces a dictionary which gives a decode for every identifier used in the body of the program. Next, we wrote a section which constructs and plants the small open subroutines (or *macros*) which perform the necessary arithmetic, logical, indexing, and transfer operations of the object program. This section accepts as a parameter of the macro the very same decode as is produced by looking up the dictionary.

When these routines had been written and thoroughly tested, we were ready to proceed to the actual task of translation. The main work is done by a set of procedures, each of which is capable of processing one of the syntactic units defined in the ALGOL 60 report. Where one syntactic unit is defined as consisting of other units, the procedure will be capable of activating other procedures, and where necessary, itself. For example, the procedure "compile arithmetic expression" must be capable of compiling the bracketed constituents of an arithmetic expression, which are themselves arithmetic expressions; this is achieved by a recursive entry to the very procedure "compile arithmetic expression" which is currently engaged on translating the whole expression.

Finally, we wrote a version of the main program. This consists of a couple of dozen instructions relating to the operating system, and ends with an entry to the procedure "compile statement". On exit from this procedure, control is transferred to the translated program, which will eventually jump back to the main program of the translator. A test is then made whether the procedure "compile statement" has been overwritten before it is entered again to read in the next program.

The decision to use recursion was not taken from an desire to use advanced programming techniques, but rather as the simplest and shortest way of getting the job done. At first, we were afraid that a coding error in a recursive procedure would be very difficult to trace and that the advantages of recursion would be outweighed by the difficulties of program check-out. I

fact, it has been found to be exceptionally easy to trace such errors, since a print-out of the push-down list gives a good indication of what has been happening in the computer. Recursive techniques may be warmly recommended in all translators which do not have to go into the most advanced methods of optimization. The greatest difficulty we have found is in providing the possibility of continuation of program checking after a syntactic error has been found in the source program. In many cases, it seems, this will be altogether impossible, and in other cases a single error will cause a large number of totally misguided error indications to be printed out subsequently. This is a problem which we are at present investigating.

Progress to Date

The following facts and figures will indicate to what extent we have fulfilled our aims.

The total length of the translator is about eight thousand instructions, and the dynamic housekeeping routines require about four hundred instructions. In combination, therefore, these occupy about half of the store of the computer. This means that the translator will not have to be re-input during the running of any set of programs which can at present be run on the standard version of the 803 computer. Furthermore, the translator can process ALGOL programs of its own size without necessitating two-pass operation. The speed of translation will be 1,000 characters of source program per second, and therefore, owing to the conciseness of ALGOL, program input will in general be faster than machine-code programs. In fact, the operating disadvantages of using ALGOL instead of machine code will have little effect on the productivity of a service installation.

I expect you will be interested in the amount of programming effort expended in the project. The coding of the translator and dynamic routines took about one-and-a-half man-years, general discussion and planning took about half a man-year, and about half a man-year has already been spent on various associated pursuits such as delivering lectures, attending meetings, working out specifications, writing manuals, conducting courses, and answering queries. The programming team consists of three people, two of whom have been engaged on the project since April 1961, and the third since November 1961. The first ALGOL program was run on 15 February 1962, and accomplished the simple task of reading two integers and printing their sum and difference.

Since that date we have successfully run programs for the solution of cubic equations, the addition and multiplication of matrices, the calculation of the Sievert integral, calculation of the greatest common divisor of two integers by recursion, the solution of linear equations, the calculation of Bessel functions, the generation of permutations, a recursive sorting method, etc.

Most of these programs were translated and ran correctly at first attempt, but some of them uncovered errors in the translator and dynamic routines. When the errors had been corrected, the programs were successfully run. In the next few months we intend to translate and run as many ALGOL programs as possible. Experience of maintaining 803 Autocode has shown that the correction of errors after publication of a translator can be a troublesome business. We are in the fortunate position of being able to ensure that most of the serious errors can be eliminated by exhaustive checking before publication of the system.

Future Developments

Although we have already started testing the translator on real live programs, this does not mean there is nothing left to be done. At present, where standard procedures are used, these have to be declared at the head of the program, which will not be necessary in the final version. Facilities for varying the format of the output are at present non-existent. Finally, the operating system does not yet allow all the facilities which are planned. These deficiencies will be removed in the next few weeks.*

The present version of the translator has been designed to provide the most convenient possible operating system for use on a minimum-configuration 503. However, possessors of a non-basic machine will not be ignored, and we will make it possible to take advantage of any extra available facilities. In the first place, the translator may be held on any available form of backing store, to reduce the time taken on re-input; in the second place, standard procedures will be provided to enable the ALGOL programmer to use the various peripheral devices.

In writing and testing the various sections of the translator, continuous use has been made of the 803 computer, which has the same instruction code as the 503. In the last few months we have been using up to seven hours a week on this machine. This has been the main factor in the achievement of a rare distinction in the field of systems programming. The Elliott ALGOL translator has been programmed, and can be proved to work, nearly six months before the availability of the prototype of the computer for which it was designed.

Acknowledgement

This report is published by kind permission of Elliott Brothers (London) Ltd.

* This forecast has proved correct.