FACT programs, others ARGUS programs, other Algebraic compiler programs, etc. Test data for the programs, the programs themselves, and parameters controlling printout of information during operation, are stored on a single tape along with the PTS System. PTS then loads one of the programs, distributes test data for the program to various tapes, inserts derails into the program, then turns the first program on. The program operates (not in the interpretive mode) up to the first derail, at which point it is interrupted to call upon other sections of PTS to print whatever is required in whatever format is desired, and then returns to the program, which goes on to the next derail. This process repeats until the program is complete, whereupon the next program is loaded, etc. This process continues in what appears to the casual observer to be a single program. The output is normally put on magnetic tape and printed later, parallel processed with some other operation. The same system is used for FACT programs except that the printouts may be provided by FACT's own report writer.

The system permits an hour's computer time to provide checkout information for a large number of programmers, and this computer time is preciously guarded. Checked out programs are placed on a master tape ready to be scheduled for production use. The Executive Scheduler is provided with information about the programs to be run during a particular schedule period. In the case of FACT programs, label information about the tapes containing the necessary files is provided to the scheduler. The scheduler checks all this information to insure that the programs can indeed run together, selects them from the program tape, puts them in machine-language form on a production-run tape, and prints out a schedule summarizing all this information in a form suitable for use at the console by the machine operator. The actual production is done under control of the Executive Monitor. This is a program which has been designed to fit less than 600 registers. It involves, of course, many overlays. Without such an Executive Monitor, efficient parallel processing would be impossible. As machines get faster and faster, more and more time is devoted to tape changes and setting up programs. With the advent of parallel processing it becomes practicable to operate several programs simultaneously, and the need for perfect organization becomes more necessary. The Executive Monitor provides for loading programs, turning them on and itself off, releasing programs that have terminated, and loading and turning on other programs, setting restart points and automatically restarting one or more programs as needed, performing the necessary rereads when a read error is discovered and it is desired to use orthotronic control to correct the read. There are several other functions performed by the Executive Monitor which are described in greater detail in our Executive Manual.

DISCUSSION

Session 1: 17 April 1962 (Morning)

The Chairman (**Mr. D. W. Hooper**, *President*, *The British Computer Society*): This morning we start another of the very successful conferences which have been held here with the co-operation of the College authorities, and I shall immediately ask Dr. Tait to open the proceedings.

Dr. J. S. Tait (*Principal of Northampton College of Advanced Technology*, *London*): There were several reasons why it gave me great pleasure to be asked by the Mathematics Department to welcome you to the College. One is that it is always thought that I am the principal of a college in a Midlands town which manufactures boots and shoes, and you can see that we are not so very far from St. Paul's. Secondly, we are glad to see you here because so many experiments in education are going on, and new universities and technical colleges are being created. By coming here you have the opportunity to see that the facilities in terms of lecture rooms, laboratories, and equipment are first class, and you already know about the stature of the academic staff. Thirdly, the spectrum of knowledge in the technical world seems to be doubling every ten years, but in mathematics it seems to be doubling every few months and it is only by having regular conferences of this kind that one can hope to keep up to date. I do not regard you as strangers. Your Committee meets here once a month and I am sure that the caretakers believe that they are members of the academic staff of the college. It is with great pleasure that I welcome you this morning. (*Applause.*)

The Chairman: There are many of you who will know more about the subject of automatic programming languages for business and science than I do myself. There are those who are here to learn and to hear what others have to say, but I hope that that will not inhibit them from making any contributions they like to the discussions. I hope that discussion will be varied and, if necessary, provocative.

Some people have believed, and still hope, that in this subject are to be found early solutions of many problems, particularly in business. Many users and potential users hope that all their program worries will soon be at an end, if they adopt the techniques about which you will be hearing. Others are more like the Frenchman who, when asked his opinion of COBOL, said: "C'est magnifique, mais ce n'est pas la guerre."

Mr. A. d'Agapeyeff then presented his paper on "Current Developments in Commercial Automatic Programming," p. 107.

Mr. R. T. Street (*Honeywell Controls*): I want to raise a point on the dynamic variable-length units. I should have thought that, in spite of the criticism and the lack of recognition of these requirements in various languages, this particular point is actually to do with the systems, in so far as the person who is using these languages should know beforehand precisely what maximum length of image is likely to be used in any system which he is to put into operation. Can the author say whether he regards that as a serious point or not?

Mr. d'Agapeyeff: Perhaps variable-length units of data are not required because the systems analyst ought to know the actual length of the data images. I think that this is not so, and that it is a mistake when it is assumed. Several firms have commercial programs in which they cannot judge what the length of the units will be, and if they are forced to give some figure, they may be wasting tape by being over-cautious about the maximum. Consider address fields—the average number of characters is probably about 20, but the figure might rise to 100. The distributors of the products of a wholesale chemist, for instance, would have thousands of items on some of their accounts, whereas the *average* number may be 10 or 20 items. It would not be sensible to operate with standardized lengths in such a case.

Mr. Street: To some extent I agree with the author, but on the other hand the difficulties which arise through specifying variable-length units probably involve slowing down the actual machine processes which are involved internally, so that costs would probably come down on the one hand and increase on the other. In my opinion, to some extent they would cancel each other out.

Mr. d'Agapeyeff: Mr. Street may be right, but we must remember the time taken in organization. Machine time for doing the calculations is likely to be negligible compared to the time taken for organization within the program. It is this that matters, and that is why, by having the kind of machines we want, we can make savings. If it takes longer to do the final bit of work, it might be a great gain in the end because of the lack of program organization required to get to that point.

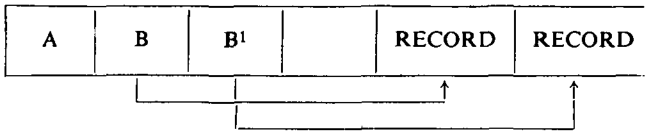
Mr. P. J. H. King (*Associated British Picture Corporation Limited*): I refer again to the need for commercial programming systems to have facilities for variable-length records. I disagree with the view expressed in the discussion that fixed-length records are satisfactory, it being the function of the systems analyst to determine the maximum record length required. This is not, in fact, always possible. In ledger processing, for example, it is often impossible to know the number of entries you might have. Even if you knew, to allocate the maximum space to each account on tape, in order to allow for this, would be absurd, as some accounts may only contain a trivial amount of data. Apart from the pure waste aspect, a further disadvantage of fixed-length record storage is that more tapes are required for a given amount of data, and hence summarization becomes more difficult.

We have recently put into practice a procedure for variable-length storage of data giving a saving by a factor of three to four over the fixed-length record procedure in terms of the amount of tape required. It also has tremendous advantages in terms of summarization. Whereas, with fixed-length records, one tape was required for each quarter and hence a comparison of two years data involved the use of eight tapes, with variable-length records one tape carries data for one year and the above comparison required only two tapes. From a system-investigation aspect there is also a saving since one is only concerned with the average record length and not the maximum possible record length, and the former is a much easier statistic to determine.

Mr. K. W. Lawrence (*G.P.O.*): Perhaps the author can clarify what he said about variable-length records. I am not clear whether he is criticizing COBOL for not dealing with variable-length records, or whether he is saying that there is some inherent problem which means that it will never have such a facility.

Mr. d'Agapeyeff: I think that Mr. Lawrence has misunderstood me. I was not criticizing COBOL for not having variable-length fields, as of course the language has these and an extra generality of levels within records. But it is here that we have the fault of present machines.

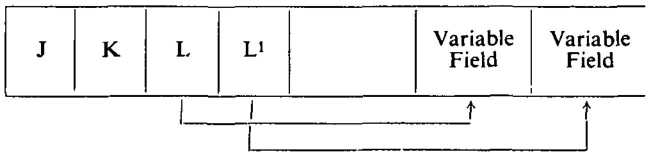
Let us consider the sort of thing which the user wants to do. He wants variable-length blocks on tapes. This makes it very difficult for a compiler to work with fixed-length blocks. The user then wants a number of records to a block. Since the position of each record is uncertain we need a map of the block (Fig. 1).



A: Number of records.
B: Record address.

Fig. 1.—Block map

Next, the user wants variable-length field inside his records. Now we do not know where each field will start so we need a map of each record (Fig. 2). Finally, he may



J: Length of fixed part.
K: Number of variable-length fields.
L: Address of variable-length fields.

Fig. 2.—Record map (part)

want variable-length fields within a list (i.e. of repeated items), in which case we do not know the starting point of each item within the list, even after evaluating the subscript, and further mapping or searching is required.

Thus, whatever you do when you finally get to the required field is of no consequence, because of all the time spent in getting there. We must look to logical designers to help us alleviate this problem.

Mr. C. Strachey (*Private Consultant*): The point about variable-length records as against fixed-length records is a rather good illustration of a general point about all compilers and in particular about all commercial compilers. It is the influence of the desire for efficiency, in the sense of having a run-time program which is fast, on what people think should be the language, or how the compiler should do its work.

It is quite clear to me that a language which has the ability to have variable-length fields as well as fixed-length fields is a very powerful language, for there is absolutely no reason to use the variable-length field unless you want to. If the compiler is such that variable-length field operation is bound to be less efficient, that is to say, it would take longer than a fixed-length field, then, in circumstances where you can tolerate the loss of space involved in using a fixed-length field, it is best to use fixed-length.

There is relatively little difficulty about this sort of operation, provided that you have enough core store in the machine. Unfortunately, practically all the machines made in this country have a totally inadequate amount of core store. This is partly the result and partly the cause of the fact that our manufacturers are so backward in the development of compilers. You will perhaps see what I mean if you consider the problems of trying to run the compiler we have been discussing, which has 220,000 three-address instructions, in a machine which has a store for only 1,000 single-address instructions. It would obviously be foolhardy in the extreme even to attempt such a task. When this is coupled with the demand that both the translater and the object program should be very efficient indeed, the problem becomes merely preposterous.

It is the insistence on trying to sell small, cheap machines to do large and complicated problems, and insisting at the same time that the resulting program should be at least 100%, if not 200%, efficient, that causes this country to be so behind in the field of compilers.

Mr. d'Agapeyeff: Of course, we do not have suitably sized machines. We do not have suitably sized anything, but that is not the main point. The main point is that machines are not properly designed. Yet, we do know the kind of order-code design that we need, as soon as we can get the salesmen and users away from it.

For example, one aspect of a translator is not difficult to implement, and is so obvious that we should have had it years ago—namely *addressless* instructions. The moment you put addresses on instructions, you cannot move them about easily, and this is why American compilers are so bad. After the first scan, a great deal is known about the source program. The next step is to pick up the so-called generators. The problem is to know where to put the generators, since if they call each other the set required at any one time is difficult to determine. This gets so bad, because you cannot slide them about dynamically, that there is a tendency to take out all the calls, sort them against the generators, and finally re-sort the generated routines back into the original sequence. But all this takes a great deal of time.

Once it is appreciated that it is organization which takes machine time, not computation, there will be a tremendous saving. I agree with Mr. Strachey that we have bad machines which are badly designed. They are not designed for the work they are meant to do, because, *inter alia*, the logical designers are not in general working with the compiler writers.

During the Session, Mr. Hooper vacated the chair and Mr. R. L. Michaelson (Vice-President of B.C.S.) took over.

The Chairman (Mr. R. L. Michaelson): It is always a pleasure to welcome an American, although, apparently, we do not all agree with all that is done in America. Dr. Clippinger is especially to be welcomed for he goes back almost to Computer Biblical times, and when one thinks of Clippinger one thinks of Eckert and Mauchley, who must be regarded as the Isaac and Jacob of computers. He first worked on ENIAC as long ago as 1945. I think that he can be regarded as the man with the greatest computer experience in this room. He is also a member of probably 90% of the committees which Mr. d'Agapeyeff mentioned earlier this morning. There is no need to say more about Dr. Clippinger. His name is his own introduction.

Dr. R. D. Clippinger then presented his paper on "The Operational Usage of Commercial Compilers in the U.S.A," p. 112.

Mr. E. Humby (*I.C.T.*): Mr. Ellis has asked me to convey his apologies for not being able to attend and present his paper on "COBOL in the United Kingdom." I am a completely inadequate substitute, but the Chairman has been kind enough to allow me to state I.C.T.'s rather special position in relation to COBOL.

In various journals we see mention of the attitude to COBOL in this country being "lukewarm." But that is a statement about a specific period of time, and does not properly convey the effect which COBOL is having in this country. I should like to rephrase that statement to read that "the acceptance of COBOL is slower in Great Britain than in the United States," and that is a rather different matter.

There are various good reasons why acceptance is slower here, and I hope to show that they are gradually dwindling away. First, capital investment in computing equipment by British users is on a much more conservative scale than is investment in the United States. There are customers who have tried to do payroll and store accounting on machines without magnetic tape, and who have expected to be provided with autocodes for programming their problems. Even where customers talk in terms of big machines, the number of tape units is always considerably lower than is the case in America, and it is this reason why some American manufacturers selling in the British market were reluctant to transfer their COBOL compilers and offer them for sale in this country—because there would be a minimum number of customers with equipment of the size necessary to cope with their compilers.

Secondly, descriptions of COBOL were very much tape-orientated, and this meant that there would be problems of defining input and output facilities for order on card or tape-punch machines.

Thirdly, there was a certain vagueness about the first COBOL report, and this meant definition problems for implementers.

Fourthly, there were the great distances which separated British implementers from the source and from the COBOL maintenance committee in the United States, which aggravated the definition problems.

Fifthly, there was no driving force in the United Kingdom equivalent to the Department of Defense of America. While it is true that the pressure which the Department of Defense applied on the American computer industry is such as to make a refusal to manufacture COBOL almost rank as an un-American activity, nonetheless, the position was sufficiently developed to make it evident that acceptance of COBOL in the United States was already widespread among customers other than the Department of Defense.

Sixthly, there are some people to whom COBOL must always be unacceptable, because of the cost of providing readability.

These reasons will have less and less effect, and as they do so the acceptance of COBOL is likely to be accelerated.

I suppose that I.C.T.'s rather special position stems from the fact that some eighteen months ago a special report by a working party of the B.C.S. declared that COBOL would make no impression in Great Britain, and that manufacturers would be advised to pursue their own languages for their own machines. We ignored that good advice because our crystal ball told us that things would be otherwise. The position is now that we will shortly be working our 1301 Mk. I COBOL compiler. It is the Mk. I because we have seen that a very special compiler must be written for machines of very small configuration, and it was our wish that we should provide COBOL facilities for all our users, including the users of the minimum configuration; the Mk. I was designed to work on programs for the machine with 400 words of immediate-access store and a backing drum of 12,000 words, with a card reader, card punch and printer, and no magnetic tapes.

Of course, we are already clear that there must be certain inefficiencies about a compiler for such small machines. For example, we can have only 100 PERFORMs in one program. That is not so serious, but there are more serious restrictions in the variety of records in one program. We are planning a Mk. II, to be developed later in 1962, to cater for magnetic-tape facilities and to make use of the larger immediate-access stores available, in order to produce a more efficient object program.

At this point I was to have tried to convince you about the *inevitability* of COBOL in Great Britain, but Mr. d'Agapeyeff has done that for me. That has surprised me, because I was under the impression that he was a member of the B.C.S. group which produced the report on COBOL about which I have just commented.

However, there is a difference in that he feels that COBOL will be foisted upon customers by wicked salesmen, whereas we think that it is the only practical common commercial language, so that COBOL will be foisted on manufacturers by the customers. It is for those reasons that we as a company are resolved to be first in the field, and not to waste effort on a language which would be outmoded before it was implemented.

Those manufacturers who took the advice they were given have not produced a common language of sufficient power seriously to threaten COBOL. We hope that our Mk. I and other efforts, such as I.B.M. on the 1401 and the smaller configurations of 1410, will convince people that COBOL compilation on small systems is a practicality.

With regard to the vagueness of reports and the distance from the maintenance committee; work on the Association of European Computer Manufacturers is in progress, aimed at resolving some of the ambiguities of the COBOL reports and establishing a more direct link with the COBOL maintenance group.

I can only ask those who find COBOL unacceptable, because of its verbosity, to keep awake after lunch long enough to hear what I have to say about the I.C.T. *Rapid-write* as a form of reading COBOL into a computer.

As these obstacles slip away, we are confident that COBOL will take a stronger and stronger grip. All Mr. d'Agapeyeff's fears of the effect of stagnation with the acceptance of COBOL on the design of computers leads me to the comment, which I can only make with acknowledgement, that the extensive production of motor-cars in no way seems to be prejudicing the development of the aircraft as a superior form of transport.

Before I conclude, I should like to make some comments on Mr. d'Agapeyeff's observation on the selling of incomplete languages for use with incomplete compilers on incomplete machines. Providing that the degrees of incompleteness are nicely balanced, this is the correct way of going about things. I recall visiting a motor-car factory when I saw incomplete carburettors and incomplete chassis and incomplete engines waiting for the time when all these things would come together and there would be a workable car. If the potential user had had the good sense to complete his driving lessons at the same time as all these things were being completed, he could have got into that completed car and driven it away. If we were to wait for a working computer before we thought about a programming language, and if we were to wait for a working compiler before we taught our customers how to use these languages, we would very soon be out of business.

Mr. R. M. Paine (*C-E-I-R-* (*U.K.*) *Ltd.*): I was intrigued and stimulated by Mr. d'Agapeyeff's talk, which was something of the combination of a windmill and a whirlwind—his fists flew in all directions with a lot of wind assisting in their propulsion.

He was worried about standardization, but I think that he will find that there are several languages growing up together for specialized aspects of the commercial and scientific world. I do not believe that COBOL or ALGOL is the complete answer. We will require specialist languages for particular uses. We may have different languages for, say, market-research problems, statistics, and invoicing on the commercial side, and for instance linear programming and matrix arithmetic for scientific studies. But I am not worried that we seem to be standardizing on COBOL. Other languages may grow from COBOL, partly because people will get too fed-up with the verbiage of COBOL. This verbiage will probably be defended by Mr. Humby on the grounds that it produces full readability for the manager to be able to understand what is happening and enable him to control the programmer. Mr. d'Agapeyeff wondered whether the manager could understand autocodes. I do not think that the manager would or should understand the full print-out of a COBOL program.

Dr. Clippinger gave one instance of the use of FACT in which he said that the statement was, "If not asterisk, go to Procedure *B*, otherwise go to Procedure *D*, until *J* equals 6." Very few managers will go into such details to understand what the programmer means. But there is no great need to produce a full English-language description of the program, provided sufficient analysis of what is required has been done previously. I do not think that managers will be concerned to that extent, with the English print-out.

I believe that most compiler writers will agree with my views, since the verbiage is completely stripped from the statement as soon as the compiler gets to work on it. Eventually, we will get closer to using symbols within any commercial language, rather than the long sentences which we have been using.

I am also worried by Dr. Clippinger's comment on the efficiency of compiling times and running times. With the new, faster, and costlier machines such as Atlas and Stretch, the inefficiencies of the plain-language autocodes will be more costly—seconds lost will mean pounds. I cannot see people, especially organizers of service bureaux, using a language such as COBOL (or FORTRAN written for the 704 but used on the 7094) which would cost them such a loss in money or in time. I think that they will produce their own specialized languages to make the best use of the large machines. There will be specialist languages for the problems that are to be run.

Mr. d'Agapeyeff criticized the manufacturers who, under the influences of the Department of Defense, had foisted the language of COBOL on the users; but I do not think that the users will ever get together and plan a language. The force must come from the manufacturers and other people such as the consultants. It is pie in the sky to think that the users will work together to find a language. Mr. d'Agapeyeff has previously made great play about the users not being consulted about COBOL. I think that he is wrong and that they were, as far as was possible, but normally they cannot say what they require in an autocode.

He seemed to be calling for a committee on the standardization of the committees on standardization, but I do not think that a committee would work in the present situation. He also appeared to be calling for a merger among British computer manufacturers, and when he did I expected a few to get up and fight back. There may be some mergers, but I would like to hear which he considers would be most useful, especially in view of the present state of compilers and the development of languages.

I did not believe him when he said that a standardized language would cause the machines to become standardized: in other words, that new machines would be very much like the old. I was startled, however, to hear Dr. Clippinger say that the 1800 had the same order-code structure as adopted in the 800, in order to make it easier to compile FACT programs. That worried me. Previously I would have thought that we had got away from the idea, as Mr. Humby put it, that aeroplane development would be damaged by the motor-car. But we may have to consider more seriously how standardized languages will affect the design of new machines.

There were a few wild comments in Mr. d'Agapeyeff's speech about computer manufacturers over-emphasizing the compatibility of a common language. But I believe that compatibility does not mean that a program can move from one computer to another without alteration; but that it is easier to learn another language of the same type. If you have already learnt the COBOL autocode as defined by one manufacturer, it is easy to learn the COBOL version of another manufacturer; thus the users can move to different machines more easily, and the programmers (a very important point) can move more easily between users and machines.

Mr. d'Agapeyeff had a great deal of fun attacking existing languages which all seemed to have certain defects, but he did not mention one with which he had a great deal to do—SEAL. I do not know what has happened to it. Perhaps he has drawn a veil over it as it was too perfect for this imperfect world.

He seemed to be asking for faster and larger and cheaper computers, using compilers with more features than any now existing but requiring only 4,000 instructions instead of 200,000, compiling faster than ever before, and running at a lower cost at a greater speed. That is completely impractical, and reminds me of the time when certain people thought that autocoding meant that all you had to do was to stand in front of the machine and say, "Do Payroll" or "Prepare Invoices" and the computer produced results.

I should like to ask Dr. Clippinger whether there is any provision for segmentation in FACT. He mentioned the size of the compiler, but the object language programs can also be very larger—too large for holding in the machine at one time. Are there any facilities to allow the programmer to state which part of the program he wants in store at any particular time?

When people have the idea that preparing an autocode program is very quick, they forget all the analysis behind it which is so very essential. This makes me believe that we should spend more effort on how to use existing machines effectively—find real applications for them—than on producing new machines or new autocodes. I understand that in the computer manufacturers, for every 1,000 hours devoted to developing better computer equipment, just one hour is spent in devising methods to improve the use and applications of present computer equipment. This is not unlike the situation in the aircraft industry in which jet airliners which are larger and which will fly faster and further than ever before are developed, while enormous losses are made with present machines.

Following on my remark about the necessary analysis behind any program, which takes longer than the programming itself, I should like to ask Dr. Clippinger whether the period of three man-months which he mentioned as being taken to produce the payroll program include the analysis? I think his answer will give us a better appreciation of improvements that may be produced by autocoding.

Dr. Clippinger: With regard to segmentation: a procedure is said to be a major procedure if it is a procedure at the top level in FACT language. There can be procedures which are *sub* to other procedures. A major procedure is automatically a segment. There is language in FACT to release a segment. When you refer to a program which is not in memory, the program is automatically loaded. After you release it, it still stays there, and if you return to it, it is still there, unless you refer to a program which causes the loading of a new procedure which would then overlay it, in which case it would have to be a rewrite. When you go to load the new procedure, it turns out that there is enough memory for it, but it is not all in one consecutive piece, in which case the segmenter loader of this program sees if the new procedure is to be brought in. This is done automatically and the programmer does not have to concern himself with it. In summary, there is a considerable amount in FACT to handle segmentation and to make it easy, since it is absolutely imperative to handle big programs on modest-size machines.

I was asked if the three man-months for the payroll application covered the system's analysis. It did not. The program had already been written in ARGUS (Honeywell's Assembly Language) and was not only completely thought out but coded. The only claim is that to do the converse process, where you work out the systems and do the program in FACT, and then decide that you want to program it in assembly language, is a bigger job. To get it in the assembly language cannot be done in the same three man-months.

Mr. d'Agapeyeff: I make no complaint about the remarks made to me; I expected a certain following up of my talk. They are a sign of the ownership of the corns on which I was treading.

We* did not say that COBOL 60 would not have any impression, at least I hope we did not. We said that we could not recommend that the language, in its then state, was such that manufacturers should give up their own projects in favour of it. Of course it is right now to say that there is no alternative to the acceptance of COBOL. But, at that time, we thought manufacturers would press ahead with the work they were doing, otherwise there was not much point in starting. In the event they did not, and if COBOL appears to have been foisted on us it is effectively the only thing possible here. Dr. Clippinger has an alternative, but we do not have much else.

Mr. Paine spoke of using a specialist language, and I agree that users have to work towards that when they have special requirements. The manufacturers will only provide a general tool, and if users find that it is not wholly suitable they will

* This refers to Editors of "A Critical Discussion of COBOL," d'Agapeyeff, *et al.*, *Annual Review in Automatic Programming*, Volume 2, Pergamon Press.

have to produce their own. He (Mr. Paine) belongs to a company which has done this, but there are few others. Users have no right to complain that they do not like what the manufacturers are offering, if they are not prepared to do anything themselves.

I was not recommending mergers among manufacturers in this country, but I think they will come. If, one way or another, we do not get more done in programming generally, we shall be entirely reliant on the Americans for all services, not just machines.

Standardization of a kind is already occurring. Last night we were asked why we should standardize on COBOL or anything else. But there is this tremendous investment in the language which, it is said, we have to use in the best way we can. Again in regard to order codes: compilers are written in a certain language, and all machine programs (e.g. service routines) are written in that language. If there was a standard order code we could never get away from it. I am not surprised that, having written 200,000 instructions, Dr. Clippinger did not want to re-write this program.

The remark about my own language was perhaps a little unfair because I am unable to answer for myself in view of the position we hold as consultants in relation to our manufacturing clients.

Mr. C. Strachey: The lecture from Dr. Clippinger and Mr. Humby's contribution seem to illustrate something very clearly which must considerably confuse anybody who does not know a great deal about the construction of compilers. Dr. Clippinger has described a compiler for FACT which, he says, uses 200,000 instructions and which is obviously enormously large and costly. Furthermore, it operates, in a complicated programming system or *COP*, whereas our machines are "no cop"! We do not have such operating systems working.

On the other hand, Mr. Humby said that he had implemented COBOL for machines with no tapes. The two statements do not go together. The ordinary person who does not know what is going on cannot understand why it should be necessary to use 200,000 instructions in America to do something, while in England apparently the same thing can be done in a much smaller number of instructions on a machine which in America would not even be regarded as an off-line printer. Clearly, there is something wrong. The truth is that in the form of COBOL which has to be implemented in the very small machines, very strict restrictions have to be made on the generality of the language, and those restrictions are generally overlooked or ignored by people who describe the COBOL translators for the small machines.

If you want to put a general programming language on a small machine, it is very important to realize the extent to which you have to throw the burden of the machine's deficiencies on the man who uses the language. It would be very much better to describe the sort of COBOL-like translators, for small machines, with a little more accuracy about the extent to which they will do what the COBOL language would be expected to do on machines in America.

Mr. Humby: I admitted that with the Mk. I compiler there were some restrictions on the use of COBOL as a language. On the Mk. II compiler where we could implement COBOL with magnetic-tape facilities but without the use of tape as such, it would be quite feasible to implement COBOL 61 in its entirety in something like 45,000 instructions. This would result in three or four passes on the 1301 with a 12,000-word drum and something like 1,200 words of immediate-access store, and it would still be a small machine in comparison with the others. If there is anything missing from this COBOL—and I do not know what it would be—I cannot see that it would be worth the effort of a colossal number of instructions. In any case we have many customers who would be prepared to make the compromise and accept the restrictions, if they are clearly outlined.

Dr. Clippinger: The efficiency of the object code was a major effort in the whole design of the FACT compiler after the language was specified, although not so much in the specification of the language. The code produced is much more efficient than it could have been, and we are making constant changes in it to make it even more efficient. Our goal is to approach as closely as possible the best which can be done so there will be no worry about the generalities which are handled by a compiler. We do not expect to achieve that, but we expect to get close enough to it so that it will become a useful tool and will be used.

For example, sorts in the FACT compiler will sort just as fast as the sorts in the ARGUS library. File maintenance, which is another major aspect of data processing, is made as fast as possible. It is buffered in such a way that the tapes are kept moving, and there are cases in which the FACT program will exceed in efficiency the program which a good programmer would put together doing it on his own, because of the great sophistication built into the input/output program, and the packing of the files and the elimination of waste space in the fields. There is no insistence on maximum-size records, for example, so the files are considerably smaller than the files which a programmer might have dreamed up for himself. He would not be willing to go to the trouble, which we went to, to squash the pile down tight. There is some inefficiency there; we are trying to remove it as much as possible, and we are making progress. I think that the end result will be judged to be good, on balance. There will be a tendency to use somewhat more memory, but I do not think that any of our competitors will do better than this. I think that it will be a good trade-off. There is so much convenience, and so much saving in money in reducing programming costs, that you can afford to spend a little more for memory. Furthermore, our memory costs are coming down rapidly so that the objections to the extra cost of having more memory are becoming less, fairly fast.

Mr. P. Wegner (*London School of Economics*): Dr. Clippinger has said that memory costs are coming down rapidly. I think that compiler writing costs are likely to come down just as rapidly. What is the ratio of the time taken to write a compiler inefficiently to that which would be required to write it efficiently? I suspect that this factor can be as much as 100, especially when we take into account the Parkinsonian inefficiencies of working in large groups. Would Dr. Clippinger agree with the point of view that the difficulties in writing compilers are transitional, and that the cost of compiler writing might well become negligible when appropriate languages for writing compilers are developed?

Mr. d'Agapeyeff stressed the desirability of having compiler-writing features in future hardware. Although I basically agree with him, I should like to point out that this is a reversal of a long-standing trend, in that it implies special-purpose hardware features; thereby negating a tacit assumption which has been underlying the development of computers—i.e. that they should be general-purpose in their hardware. Ten years ago, when computers were first being built, the emphasis was on general-purposeness. The guiding principle was the common store for instructions and data—as outlined by Von Neumann. Persons who did not swim with the tide, and advocated separation of data and instructions, were branded as reactionaries. Today we seem to be coming back to the idea of special-purpose features and special stores for different categories of data and instructions. For instance, the Atlas has four different memories for different categories of information, and machines with lock-out bits or memory limit registers effectively use these to differentiate between categories of memory. Before we advocate hardware to implement compilers, we should be clear that we are advocating more special-purpose features. The question is whether that is entirely a good thing. I probably think it is. After all, the most general machine is a Turing machine, and nobody complains about arithmetic and even floating-point facilities. However, I think that we should be clear about the implication of advocating special-purpose features.

Dr. Clippinger: If we built FACT again, we could do it for fewer man-years. I doubt whether the factor is as high as 100. I believe that there is an irreducible minimum involved in something as complex as FACT and that we might get the figure down to ten man-years. If we did, it is an interesting exercise to extrapolate in order to estimate the time which might be taken for the tenth edition of FACT!

In the process of doing this, we are learning about compilers and the better ways of handling them, and how to get the job done for fewer man-years. But this has no effect on the end result, which is what the programmer is using. That point should be kept clearly in mind.

Mr. d'Agapeyeff: I do not believe that compiler readability will lead to special-purpose machines. It is a general-purpose tool although it can have a wide variety of uses for many other things and certainly we have used it for purposes which were never thought of originally. Special-purpose input/output or the use of interpretative or semi-interpretative techniques will require better machine design.

The Deputy-Chairman: There is no more time for questions, but if anyone has anything more to say, he will not have much difficulty about re-phrasing his question to make it appropriate to another session. (*Laughter.*) I am sure that you would not like the session to conclude without thanking the three speakers who have given our conference such a good start. (*Applause.*)

The Conference Adjourned.

Operating experience with ALGOL 60*

By E. W. Dijkstra

This paper describes the circumstances under which the ALGOL 60 translator of the Mathematical Centre, Amsterdam, has been constructed. It describes its main features, virtues and shortcomings. Furthermore, it tells how the translator has shown itself to be an inspiring tool for a varied group of users.

For the Computation Department of the Mathematical Centre, Amsterdam, ALGOL 60 came exactly at the right moment, i.e. some months before our new machine, an X1 produced by N.V. Electrologica, The Hague, came into operation. And before going on I should like to mention some of the circumstances which greatly eased the development of our ALGOL 60 translator.

(*a*) As our older machine, the ARMAC, was still in operation and taking care of the service computations as in the preceding years, we could give the development of our translator the highest priority as far as X1 machine time was concerned.

(*b*) Faced with a new machine we were relatively free from preconceived notions as to how this machine should be used. No traditions with regard to its use had to be broken, because such traditions had not yet grown.

(*c*) As the speed of the X1 was considerably higher than that of the ARMAC, we had the joyous feeling that the speed of the object program did not matter too much. The resulting frivolousness saved us a great amount of trouble and pain.

(*d*) The fact that the X1 is a fixed-point binary computer enabled us to include suitable red-tape operations in the floating-point subroutines at little or no expense as far as run-time speed was concerned.

(*e*) Thanks to the fact that our previous machinery was highly unsuitable for any form of automatic programming, we did not have the slightest experience in language translation. We thought that this was a great drawback; it turned out to be one of our greatest advantages. Again there were no obsolete traditions to get rid of.

(*f*) In contrast to the great speed of the X1 its store was small: 4,096 words of ferrite-core store and no backing store. As a result all "strategy questions" where "space" and "time" had to be weighted against one another were settled almost automatically.

(*g*) As some of us, in particular Prof. A. van Wijngaarden, had been heavily involved in the creation of ALGOL 60, our group, as a whole, was probably more tolerant with respect to its less lucky features than would otherwise have been the case. As a result we did not waste our time on discussions as to subsets or modified versions, but took the challenge as it stood.

On the other hand, the limited size of the store has had two undesirable effects. Firstly, we did not dare to try a "load-and-go translator." We therefore aimed at a single-pass sequential translation, simultaneously reading the source text and punching out the object program. (It turned out that this pass had to be preceded by a rapid so-called "prescan" in which identifiers of procedures and labels are collected.)

* Report MR47 of the Computation Department of the Mathematical Centre, 2de Boerhaavestraat 49, Amsterdam, Netherlands.