By R. F. Clippinger

This paper, on the operational usage of the commercial compiler FACT, was presented at the conference under the original title "Operational usage of commercial compilers in the U.S.A."

FACT was described and compared with COBOL and Commercial Translator in the Annual Review in Automatic Programming, Volume 2, Pergamon Press, 1961. For that reason, the description of the language contained in this paper will be illustrative and drastically shortened.

The plan to create FACT took form in January of 1959 and work started in May, just a month before the start of the development of COBOL. The development was, therefore, during the crucial first months, independent of COBOL. At this time, it became Honeywell's purpose to provide the customer with the best dataprocessing tool that we could conceive. The design of a language and the corresponding compiler requires a long series of judgments based on extrapolations of expected consequences of a course of action. We do not claim that our judgment is correct on every point nor even on overall balance. We do claim, however, that FACT is a significant data-processing tool and that many of our judgments have influenced the other representatives of the COBOL Committee to incorporate features into COBOL which otherwise might not have been there or might have been there in a different form. This influence is still being felt. A SORT verb and a REPORT WRITER will be part of COBOL 61 extended. Honeywell played an important part in each of these groups. Other features of FACT are likely to find their way into COBOL 63. In line with our plan to make FACT as useful as possible, we tried to make it sufficiently complete so that it would be possible to do whole dataprocessing applications in the language of FACT without ever leaving it for a lower-level language. It is, thus, the first data-processing compiler to give attention to easy and extensive editing of information coming in from and going to punched cards, the first to provide a flexible means of handling information coming from and going to paper tape, the first to have built-in sorting, and the first to consider the manipulation of information io and from magnetic-tape files so important that the housekeeping of this manipulation was built into the compiler. It is, in summary, the most extensive dataprocessing compiler in existence and is more extensive than any other compiler on the drawing boards. FACT permits the use of an assembly language, ARGUS, to handle other data-processing peripheral units such as bulk storage, inquiry units, and transmission lines, but even FACT has not handled these units in compiler language.

The goals of a data-processing compiler are:

1. To reduce problem preparation costs.
2. To reduce the number of man-years of effort required to get data-processing jobs ready to go.
3. To reduce the effort required to modify a dataprocessing program to reflect changing requirements.
4. To reduce elapsed time between problem definition and working program.
5. To improve documentation of the job being done.

It is too soon to have a good measure of the success of data-processing compilers in meeting these goals. FLOW-MATIC, the first data-processing compiler, never received sufficiently extensive use to provide a decisive answer. COBOL, despite the impressive number of manufacturers implementing it, and despite the fact that one manufacturer has had a compiler in customer's hands for over a year, has still no significant set of users with complete applications running; experience is still limited with FACT and Commercial Translator. A few qualitative remarks can and will be made in the course of this paper, but the real volume of use on complete applications, which will provide a true measure of the usefulness of data-processing compilers, is still a year or more away. It is, therefore, too soon to make any judgments, and this paper should be considered merely a discussion of the reasons for which we, who have pioneered in the use of data-processing compilers, feel confident that we will meet the goals, and that customers, after some initial unfavourable reactions, will eventually learn to see data-processing compilers in their proper perspective, know when and how to use them, and what complements of equipment to acquire for their effective use. We predict, in summary, that the trend towards the use of data-processing compilers will continue and will have an important influence on machine rentals and machine design.

## Description of the Language

Complete descriptions of the FACT language are available in FACT manuals, which are available to interested persons. Less complete descriptions are available in the above-mentioned article in the Annual Review, or in the FACT primer.

The key to understanding FACT is the understanding of the structure and definition of a file on magnetic tape (see Fig. 1). A file is given a name (STOCK TRANS-
fle outline form


Fig. 1.-File Outline Form

FER) and contains certain "secondary groups" (COMPANY, STOCK-HOLDER, CERTIFICATE) of fields (CO. NO., STREET, DATE-ISSUĖD). The file is described on a form, as in Fig. 1, whose lines are keypunched in cards called FILE OUTLINE DESCRIPTORS, labelled 0 in the first column. Various groups of columns on this form permit the identification of the descriptor and definition of field parameters, such as LENGTH, MODE, DECIMAL POINT. The fact that data-processing compilers in general allow the use of fields whose length is independent of machine wordlength is one of the many fundamental differences between a data-processing compiler and a scientific compiler, such as an ALGOL compiler, and is one of the many reasons why a data-processing compiler tends to be much more complex than a scientific compiler. The secondary groups, which have an asterisk before their name, belong to other secondary groups and contain secondary groups; thus, STOCKHOLDER belongs to COMPANY and contains CERTIFICATES. Any number of CERTIFICATES may be contained in one STOCKHOLDER, any number of STOCKHOLDERS in one COMPANY. The fact that STOCKHOLDER belongs to COMPANY is represented by the indentation of STOCKHOLDER with respect to COMPANY. The
language of FACT provides verbs, such as GET, OPEN, FILE, and FILE ENTIRE (which should be considered a single verb) which operate on the nouns (e.g. STOCKHOLDER). A statement in the middle of a FACT program, such as GET COMPANY, causes the compiler to generate an input-output program which brings in from magnetic tape to memory one COMPANY header (that is, CO-NO, COMPANY-NAME), one STOCKHOLDER header, and one CERTIFICATE, making this information available so that subsequent statements referring to some of their fields, such as NAME, can be executed. Other data-processing compilers, such as FLOW-MATIC, COBOL, and Commercial Translator, require the programmer to do his own housekeeping using more elementary statements to handle COMPANY, STOCKHOLDER, and CERTIFICATE separately. Such an innocuous-appearing FACT statement as FILE ENTIRE COMPANY leads to the insertion of an input-output program, whose execution brings in from an input file and files out to an output file a succession of all CERTIFICATES belonging to a given STOCKHOLDER, and then a STOCKHOLDER header and all his CERTIFICATES, and then another STOCKHOLDER and all his CERTIFICATES, etc., until there are no more STOCKHOLDERS, and then,


Fig. 2.-Card Descriptor Form
finally, a STOCKHOLDER summary. If this is a reversible file, it then puts a STOCKHOLDER header at the end of each STOCKHOLDER group and a COMPANY header at the end of the COMPANY group so that the file can be read in either direction. This corresponds to a fairly large source program in COBOL. Because of differences like this between FACT and other data-processing compilers, such statements as "Compiler A compiles N source statements per minute and Compiler B compiles M source statements per minute" are largely meaningless. The file outline constitutes the basic document to the FACT compiler, providing the identification of the bulk of the words used in the source program.

## Card Input

The information contained on cards, which will be fed to the object program, is described to FACT on the form shown in Fig. 2. The field names used are the same as the names used on the file outline. There is provision on the form for describing where the field is located on the card, its mode. the handling of signs and decimal point, and the checking which the programmer deems appropriate. This provision for checking is quite elaborate. We feel that this is appropriate, because the quality of the results obtained from a data-processing
object program depends on the correctness of the input information. The biggest source of error, of course. after the program is working, is the human operating the key punch. An exhaustive study of the error actions permitted is necessary to appreciate the lengths to which we have gone to purge this input data of bad information. Just a few of these actions are the following.

In column 35, one can specify a mode error action. Input fields may have any one of fourteen different modes and, if the incoming field does not belong to the class that its mode indicates, this is a mode error. For example, a Hollerith field may have any one of the 64 legitimate characters, whereas an alphanumeric field may only have one of the 36 digits and letters of the alphabet. In passing through the reader, the card is read twice and, if there is a discrepancy, this is an error over which the programmer is given some control by use in his source program of the expression "IF CARD ERROR." In addition to this kind of routine check, which he may use or not use, the programmer may have some reason to perform his own checks. For example. a certain field may be valid if it is 1,3 , or 17 . and otherwise invalid. To handle this situation. FACT provides an acceptance condition name in columns 66 through 80. This is the name of a statement in his source program

FACT
REPORT DESCRIPTION FCRM


Fig. 3.-Report Description Form
which is interpreted as a condition whose truth permits the acceptance of this field.
Even more elaborate checking or input-data purging can be accomplished through the use of the procedure name in columns 51 through 65 . This is a source program procedure, written by the programmer, which is executed at the time this card is edited and could cause the formation of some new field, which is any function of this and several other fields. This extensive programmer control of input editing of information from cards within the framework of the compiler language is another example of a feature directed at the core of dataprocessing problems, which is not yet available in any other compiler language. The Input-Editor is used by inserting into the source language program a statement of the form, see INPUT-EDITOR. The compiler reads the card descriptors, analyses them, using information from the file outline, if necessary, and source statements if the cards involve acceptance conditions and procedures, and generates a program which accomplishes the purpose of reading cards in the object program, editing the information as required, including changing it from sixbit to four-bit form if numeric, grouping it into groups, and filing these groups into a file on magnetic tape in proper FACT format. In the case that the programmer
used no acceptance conditions or procedures, the mere description of his cards and file outline of his corresponding file suffice, therefore, to provide an object program which handles all the housekeeping of fetching the cards and creating the file. It is hard to imagine how the job could be made any easier.

## Reports

Another major aspect of data processing is the creation of reports. In FACT, a report may be formed from information in a file using a source program which is stripped down to almost nothing but the fetching of records from the file and the statement WRITE REPORT. The bulk of the preparation consists in filling out forms, such as Fig. 3. These name the report, the different kinds of lines that can appear in the report, the fields in those lines, and provide the parameters which will determine where those fields appear on the line and in what form. They also provide for the creation of cross-footing information, zero suppression, the creation of tab fields and their arrangement on the printed page, and the naming of conditions and procedures which will determine when to create tab totals, whether to print a line at all or not. They permit the programmer, if he wishes,

## SOURCE PROGRAM STATEMENT FORM



Fig. 4.-Source Program Statement Form
to interrupt the output editing function with source statements of his own which may provide output which is a function of information coming from any different source. Roughly speaking, this process is the inverse of the input editing function. There is provision for creating this report on line or on tape for later off-line operation. It is important to note that the information to create a report can be obtained from many files, and that the report can be created while other processing such as file maintenance is being carried out, or other reports are being created. This is a significant improvement over report writers which require a separate pass to create a special tape for the report.

## Source Language

FACT source language is a set of words which look like English. Some of these were defined on file outlines, and others are reserved lexicon words of special meaning (SEE, CLOSE, GET, REWIND, SORT, ADD, etc.). These words are arranged into sentences and the sentences into paragraphs, named or not named, and the paragraphs into procedures (see Fig. 4). These sentences may be simple declarative sentences such as "ADD ONE TO SUM" or they may be complex, containing conditional clauses such as "IF (NUM)TH CHARAC. TER OF NAME IS NOT . . *. . GO TO CODE COMP.

OTHERWISE DO CODE FAB UNTIL J IS 6." Aside from provision for special features such as input editing and report writing (which have already been mentioned) and sort and update, there is provision for most of the operations available in other data-processing languages like COBOL and Commercial Translator. These include character, field and group manipulation, arithmetic and formula evaluation, subscripting by the use of ordinals. The use of dynamic abbreviations makes convenient the use of subroutines. Arithmetic, which has scaling problems on a fixed-point machine, is kept under mathematical control by the use of validity checking, which applies also to the moving of fields of one size to fields of another size. This is handled by statements of the form: IF VALID, PUT $A$ INTO $B$, OTHERWISE DO $E R R O R$, or: IF VALID SET $C$ EQUAL TO $(A+B) /(A-B)^{*}\left(A-B^{*} B\right)$, OTHERWISE DO ERROR. Tables may be inserted in the narrative and referenced by such phrases: E-BONDVALUE of 1926, which refers to the tabular value taken from a bond table in which E-BOND-VALUE is tabulated as a function of DATE. FACT automatically treats procedures at the highest level as segments. These segments are automatically loaded as they are referred to. The programmer has control over the use of memory by programs by the use of a phrase RELEASE

PAYROLL PROCEDURE which automatically makes available for other segments space that was formerly occupied by payroll procedure. Space and time do not permit a more detailed description of the FACT language.

## Construction and Vital Statistics of the FACT Compiler

The size of a compiler, in terms of machine words, depends principally on the complexity of the language and the nature of the machine used for the compiler. ALGOL compilers range from 2 to 20,000 machine words, depending on whether all features of ALGOL are implemented and whether a large or a small machine is used for the compiler. Data-processing compilers are larger. For example, FLOW-MATIC, the first data-processing compiler, required about 70,000 instructions on the Univac machine. COBOL and Commercial Translator processors are considerably more complex than FLOW-MATIC and range from 40,000 to 300,000 machine words.

FACT is the most ambitious DP language there is, and has been written for a machine with 4,000 words of memory, four magnetic tapes, and one card reader. The program consists of about 223,000 three-address instructions, which is the equivalent to twice that many one-address instructions. Clearly, if we had created the compiler for a 32,000 -word machine and allowed ourselves as many tape units as we like, together with facilities for bulk storage, which are available, the program could have been much smaller and compilation much faster. However, we wished to serve the customer with a modest installation, and this made our job more difficult. In the environment in which FACT was created, the program had to be segmented into eight phases, with sorts between the phases so that the data would be presented in a convenient order. Roughly speaking, 65,000 of the 223,000 words of FACT are the coding for these eight phases. The other 158,000 represent what is called Sample Programs from which pieces are selected and into which parameters are fed to create object programs.

Oversimplified, the compiler works as follows: In phase 1 , the sequence of the card descriptors is checked and tape images are created. If they are not in order, they are sorted and then, in phase 2, file outline items are formed and given. "Dewey" numbers which determine hierarchical arrangement of file outline information. The items are then sorted to alphabetic order and, in phase 3, Dewey numbers from file outline items move to other items with the same alphabetic name. Lexicon words are identified. The items are now sorted to the original order and then, in phase 4, the Dewey numbers are used to replace compound names consisting of a noun modified by an adjective, such as WAREHOUSE PART NUMBER, by a single item with a single Dewey number. The items are now resorted to Dewey-number order and, in phase 5 , packing information is generated which makes it possible to work with partial word and multi-word fields. Items are created from card
descriptors with mode words and ARGUS symbols. Reports and group moves are handled differently by putting such items on a different tape, processing them, and then merging them with the first. The items are again sorted to the original order and then, in phase 6 , the narrative is scanned and expanded to quadruples. Thus the sentence: "If A equals B and is greater than C and unless $D$ is less than or $E$ is greater than $F$; add $A$ and $B$ of $C$ to $D$ and to $E$." is expanded to: "(If $A$ equals $B$ ) and (if $A$ is greater than $C$ ) and (unless $D$ is less than $F$ ) or (unless $E$ is greater than $F$ ), add $A$ of $C$ to $D$ and add $B$ of $C$ to $D$ and add $A$ of $C$ to $E$ and add B of C to E." This is in preparation to the creation of calls to the special generator processor of phase 7. These calls involve the creation of lists or parameters and the use of a special generator processor language which contains such expressions as LOAD OCTAL VALUE, SKIP IF EQUAL, etc. Phase 7, the generator processor, then selects code pieces from sample programs in the library and inserts parameters to create pieces of assembly language (ARGUS) object program. In phase 8 , other input required by ARGIJS assembly, such as reference tables, index register loads, etc., are created.

In every phase of FACT compilation, diagnostic messages are created as appropriate, so that the output from the FACT compilation run consists of a listing of the source language statements interspersed with diagnostic comments, and an ARGUS program listing. The 158,000 instructions in sample program library are distributed approximately as follows: 30,000 in the input editor, 6,000 in the report writer, 20,000 in the $\mathrm{I}-\mathrm{O}$ package, 20,000 in the sort generator, 5,000 in the arithmetic generator, 11,000 for field and group moves, 3,000 for updating, 8,000 for files, and 55,000 for everything else.

A program as complex as FACT takes many manyears to accomplish and it is difficult to define when the job is done. Approximately 60 man-years have gone into FACT so far, and FACT has been in customers' hands in field test for about a year. The errors have not yet all been removed from it, and experience on FORTRAN indicates that we will never know when they are all gone. Compilers of this complexity are never good enough to satisfy the designers when they first start operating customers' programs. Therefore, in addition to changes that have to be made to remove errors in logic as they are discovered, changes are made to improve the compiler, both in operating speed and the use of memory, and to improve the object code produced. Compilation time for a FACT program depends on the size and nature of the program. A program of fairly typical size may involve about 200 descriptors, generate about 7,000 words of object program and take 20 minutes to compile. It will use somewhat more memory than if a good programmer had written it in assembly language without worrying about the concerns that other programmers have with all their programs.

Since FACT first went into system test, the report writer has been improved so that it handles zero suppression better. The input editor has been improved to speed it up and make it take less memory. Improvements are being instituted in the input-output package. Sorts are being redesigned to go faster. The executive loader has been improved so that segments in the object program are loaded more efficiently and parallel processing is accomplished more efficiently. A move to streamline the connection between the last pass of FACT and the first passes of ARGUS will remove several passes from the compiler. All in all, a program which takes 20 minutes to compile, a few months from now will take 13 minutes to compile. A program that will compile in 13 minutes on the Honeywell 800 will compile in about 6 minutes on the 1800 because of the greater operating speed of the central processor and tapes of the 1800 .

Most customers of FACT programs ave 8,000 words of memory or more, and eight tapes or more. There are a series of things that can be done to the FACT compiler to make it use an 8,000 -word memory more efficiently. Some of these will eliminate passes of the compiler, others will accelerate the sorts by using more tapes. If an 1800 with a 32,000 -word of memory and 16 tapes were used, FACT could probably be rebuilt to compile the program, which now takes 20 minutes, in one minute. It is clear, then, that the job of the compiler writer is never done, and it is a matter of economic balance how far he should go to improve it. It is also clear that the user cannot expect miracles of the compiler. If the compiler achieves, at least to some extent, the goals mentioned at the beginning of this paper, this is not accomplished without some effort, and the data-processing user must expect to require a system with somewhat more memory and tape to get a good job done.

## Experience with the use of FACT

Five FACT customers are writing all their applications using FACT. These are mostly fairly large applications which, when they are all working, will average 200,000 or 300,000 instructions of object code per computer. There are certain differences between these applications which are worth noting.

One of these customers is attempting integrated dataprocessing on a very ambitious scale. Input is received via communication lines on paper tape from all over the country. It is converted to paper tape images on magnetic tape, and sorted as undigested messages. The sort accomplishes grouping necessary to serve as input to many different applications. The application is so extensive that there are 400 runs per day involving 1,500 tape changes. FACF provides for automatic label checking and writing based on information supplied at schedule time. This application calls for a special program to be written to handle file maintenance of the tape labels.

Another new installation involves 50 programs, of which 11 are sorts. Some of the input comes from an IBM 1401 on 729 II tapes, and some of the output goes on 729 II tapes to an IBM 705, which was purchased. Four of these 50 programs are file maintenance; about 13 are input editing programs, and 20 are report writing programs. Checkout progress is accelerating, but the application is not yet running as a whole.

A third installation is of interest because the customer selected our system on the basis that FACT is the only compiler that permits the fiexible use of paper tape and of relatively untrained personnel.
A fourth computer is located at a bank. Their applications are in deposit accounting, loans, factory. and corporate trust. Deposit accounting is the farthest along. Some of the programs in this application are running already.

A fifth computer is being used in a military installation for several different applications, of which one is a large inventory control procedure. The processing is com-plex-the main file-maintenance run involves 150 transaction types.

In addition to these five, all FACT customers, another dozen Honeywell customers use FACT to differing degrees. One of these customers wrote many of its applications in ARGUS. Two programmers took a payroll application, starting in January 1962, and have worked on it part-time for three months. They have now working a system of six computer runs, including two sorts and one update. These runs process a 3,500 man payroll, with 5,000 daily transactions, in 25 minutes. Another 12 runs are also working. They are weekly runs involving an update of the payroll master file, printing of pay checks, check register, payroll, summary, etc. Six of these runs are concerned with the weekly incentive performance system. The 18 programs involved about 3,500 descriptors and generated about 45,000 instructions. One of the two programmers who have accomplished this is getting so experienced that he frequently writes a program, compiles it, and operates it correctly the first time. He feels that if a thorough desk check is made of the program, it should not take more than two compilation runs to make it work. He has learned that it pays to keep the narrative statements relatively simple, because then they contain less errors and are checked out faster. He says that without FACT it would have been impossible to accomplish the job that has been done since January, and expects that by June the complete payroll application, including the monthly and weekly runs, will be operational.

FACT is not the only program tool provided with the Honeywell 800 and 1800 . A series of other programs are lumped together in our advertising under the name of COP, Computer Optimization Package. Time and space prevent discussing them all, but two of them are crucial to FACT programs. Checkout is accomplished under control of the PTS Monitor (Program Test System). This system is designed to permit the batch checkout of a series of programs, some of which may be

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.

## 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.)

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.

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 ís using these languages should know beforehand precisely what maximum length of image is likely to

