The crystallographers' friend

By Judith C. Matthewman*

The system of programs which is described has been developed in order to provide a "computing service within the computing service" on the Titan computer at Cambridge University. The arithmetic involved is for a particular department with special problems, but it is hoped that the discussion of the needs of such a system and the ideas developed for meeting these needs will be of general interest to other programmers with the task of writing a related set of standard programs.

1. Introduction

Amongst the many scientific applications of computers, crystallography has a special place. There is a vast amount of work which can be helped by a computer, and much of present-day crystallography could not be done without one. Most of the computation involved is routine, in the sense that a set of sufficiently general programs should be able to handle data from any crystal being studied.

In the past, it has been unusual for all the necessary standard programs to be written by the same person, or even by several people with a view to making them compatible. This *ad hoc* approach has usually resulted from lack of time or planning. Programs have often been written by research students who are working on particular problems, rather than writing for a generally useful co-ordinated library.

With the advent at Cambridge of a new large and fast computer, the Titan, it became essential to write a completely new set of crystallographic programs, and an attempt has been made to create a unified system capable of doing all the routine calculations required by a crystallographer. It is evidently desirable that the crystallographer (who will hereafter be called the *user*) should be able to state his requirements in a language which resembles his own as closely as possible, in much the same way as a programmer wishes to write his program in a language allied to the problem he is trying to solve. The prime consideration throughout the system of programs is ease of use. The invention of a completely new "Autocode" language was thought to be too ambitious, and a very simple language has been developed. The system was named the "Crystallographers' Friend" at an early stage in its development, and this was rapidly abbreviated to "Friend".

This paper describes the needs of such a system, and an attempt to provide for them. A large part of the programming involved deals with the arithmetic, and various numerical techniques for tackling it, but we are not concerned with that part here. The paper is not directed towards crystallographers, and, indeed, crystallography will be avoided wherever possible. It is hoped that the ideas of the system will be applicable to other fields with similar requirements.

2. Requirements of the system

The requirements of a large system of programs written for others to use are:

(*a*) To shield the users from the need to understand any technical details of the computer or of the programs. Users are assumed to know nothing about programming. It is desirable that they should regard the computer as a tool, which must be used with care and forethought, but whose inner workings they need not understand. They must understand in some detail what the programs do, without knowing how they do it.

(*b*) To enable the users to carry out all accepted standard calculations easily, quickly and without error. The system must be easy to grasp and use, and the instructions for its use should be expressible concisely and clearly. The system should cater for visitors from other laboratories, and indeed other countries, who have been used to their own computing facilities. Such users may be in the process of refining the structure of a crystal, and wish to continue using Titan with a minimum of effort.

(*c*) To enable the users to provide unchanging data once only, and to remember their data for as long as necessary. This is particularly relevant to crystallography, in which the determination and refinement of a crystal structure may take months or even years, and much of the data for a particular computer run are identical with data for the previous run. Users would like to supply only those items of data which they are changing from the previous run. They would also like to be able to obey several programs in sequence in one run.

(*d*) To detect and explain to the users all their obvious errors.

(*e*) To release the system for use complete in itself, but open-ended. It should be possible to add new ideas and techniques, when necessary, with a minimum of disturbance.

(*f*) To leave the system so that it can be understood and maintained by other programmers: too many large programs develop inexplicable faults when their authors have moved elsewhere. It is also an advantage if programs have been written in such a transparent way that

* *Crystallographic Laboratory, Cavendish Laboratory, Free School Lane, Cambridge,* and *University Mathematical Laboratory, Corn Exchange Street, Cambridge.*

if an error is observed it can easily be corrected, or if some new numerical method is developed it can be added to the system.

3. Programming language used for the system

In view of (*e*) and (*f*) above, the ideal would have been to program in some powerful symbolic language. It was a matter of urgency to provide a set of programs for Titan very quickly, and such a language would have made this possible. The resulting inefficiencies in compiled programs could probably have been removed gradually by the judicious insertion of sections of machine-code program. Much of the preliminary work was written in CPL (Barron *et al.*, 1963), but it soon became apparent that a CPL compiler would not be available on Titan in time. It is, however, still envisaged that the completed system might be translated (by hand) back into CPL if only for documentation purposes.

If the system had been written in FORTRAN, it could have been transferred to other computers with FORTRAN compilers, with evident advantages. Indeed many of the arithmetic routines involved have existed for several years in FORTRAN, and some of these could probably have been adapted to fit the system. Unfortunately, at the time of writing, Titan can only be programmed in an Autocode (which is not a sufficiently powerful language for this type of programming) or its machine code, IIT ("Intermediate Input for Titan"). Although this machine code is admirably suited to the detailed logical work involved, it makes it impossible to publish any of the program itself. A disadvantage of the present system written in IIT is that it cannot be run on any other computer (including no doubt any machine which may eventually replace Titan).

4. Conventional method of presentation of data

Crystallographic data for a computer program is of two categories. The numbers which the user measures, either by eye from a photograph or automatically using a counting device, are his "observations". They are simply a set of numbers indexed over a three-dimensional array, and the only problem they are likely to present is one of quantity. Provided that the computer has sufficient storage to accommodate these observations they can be handled; in the present set of programs they are held on magnetic tape.

All the other data for a crystal consists of isolated, unrelated numbers, and has in the past been grouped together on to a single data document called, for historical reasons, a "preface". It includes such unchanging items as the type of apparatus used to measure the observations, the symmetry possessed by the crystal, its overall dimensions, and the dimensions of the tiny unit cell whose repeated presence causes the diffraction of X-rays. It also includes more transitory data such as suggested positions of atoms within the unit cell, or how many cycles of refinement are required.

	Comments
1·23 2·34 3·45 90 90 90	*a*, *b* and *c*, the lengths of unit cell sides Alpha, beta and gamma, the angles between them
4	Maximum bond to be printed
0	Maximum bond for which angles between bonds are to be printed
2	No. of symmetry operators to be given
3	No. of atomic positions to be given
3	No. of these positions to be considered as source atoms
(*x*, *y*, *z*) (1/2 + *x*, 1/2 + *y*, *z*)	Symmetry operators defining space group of crystal
Osmium 0·25 0 0·26 Oxygen1 0·35 0 0 Oxygen2 0 0·85 0·25	List of labels and *x*, *y* and *z* co-ordinates of the three atoms

Fig. 1.—A conventional preface

In a typical run using an old-style standard program, two data documents would be provided for input: an "observations" document that is unlikely to change from run to run, and a "preface" which is given in a format demanded by the program, and which contains data of the kinds described above. For any particular program the order in which items of data occur on the preface is not obvious; it is laid down in the specification of the program, and must, of course, be followed exactly by the user.

Confusion frequently arises in the use of conventional programs. Experience indicates that it is difficult to use a program which has several options requiring the presence or absence of particular items of data. Users tend to assume knowledge of the correct ordering of items without consulting the specification every time they use the program.

On a computer like Titan, run on a closed-shop basis, trivial errors are very time-consuming. If a user or programmer is in direct contact with the computer, he can correct any trivial errors on his paper tapes or cards as soon as they are detected, but if he has to wait for a run to return to him from an operating system it can take several days before anything useful emerges. It is therefore desirable that programs should be easy to use, and that the content of data tapes should be as clear to the user as possible.

5. Presentation of data to the Friend

In order to explain the ideas involved, let us examine a very simple preface, as it would be given to a conventional program, and as it would be given to the Friend.

A conventional "Bond lengths" program (chosen as an example because it is small and does not involve the user's "observations" at all) would take a preface as in **Fig. 1**; the right-hand side is explanatory comment, and would not appear on the preface.

A comparable preface for the Friend would be of the form shown in **Fig. 2.**

That is to say, if the user were calculating bonds for a crystal which he had not previously introduced to the

system his preface would be as in Fig. 2. In this preface all the data which would be required by the arithmetic routine called "Bond lengths" have been listed, and the preface is comparable with, though longer than that in Fig. 1. (The symmetry operators there correspond to a C Face centred lattice, which accounts for the difference in their presentation; in Fig. 2 we could perfectly well write

No. of symmetry operators = 2
Symmetry operators
(x, y, z) $(1/2 + x, 1/2 + y, z)$.)

It is, however, much more likely that the user has been working on the crystal for some time, using the different routines of the system. If his crystal has been given a title, such as "Osmium tetroxide", and its unchanging data (unit cell dimensions, symmetry and latest estimates of atomic positions) have been given at some earlier run and kept on magnetic tape, then a sample preface would be:

Osmium tetroxide
Bond lengths
Maximum bond = 4
Stop

Here the advantages over the old-style preface are more evident. Indeed, the Bond lengths routine could assume that the maximum bond were 4 unless told otherwise; the "Stop" is redundant as the end of the preface becomes obvious; "Bonds" is a synonym for "Bond lengths" (see §10), and a crystal name may be as small as we please.

We have thus reduced our preface to, say,

OsO4
Bonds

which would perform the calculation required, and is less prone to error than those in Fig. 1 or Fig. 2.

6. Names

In order to interpret a preface such as that given above, we define the following:

The preface is composed of a sequence of *instructions*, such as

"$b = 2{\cdot}34$"
"No. of atoms = 3"
"Bond lengths"

Each instruction has a *left hand* and a (possibly empty) *right hand* part. The left hand part is always a *name*. A name is defined to be any sequence of symbols terminated by either an = or a newline. Thus in the examples given typical names are "Bond lengths", "$a =$", "Osmium tetroxide" and "C Face centred".

When the Friend reads the user's preface, a list of all possible names which it can expect has already been read and stored. The method of storage and subsequent recognition is an adaptation of that used in the Cambridge CPL compiler (Hext, 1965). Each name, however

Bond lengths
Maximum bond = 4
$a = 1{\cdot}23$ $b = 2{\cdot}34$ $c = 3{\cdot}45$
Alpha = 90 Beta = 90 Gamma = 90

No. of atoms = 3
Atoms

Osmium	0·25	0	0·26
Oxygen1	0·35	0	0
Oxygen2	0	0·85	0·25

Lattice = C Face centred

Stop

Fig. 2.—The same data as given to the Friend

long, can be held in one 24-bit half-word of store by "scrambling" it. One character takes 7 bits, giving 3 characters per half-word; if these characters were being stored with the intention of printing the name out again, the various half-words would be stored end to end, and a long name could take up several words of store. However, when the only property required is that the name be recognized when it is read again, it is sufficient to *add* together the various half-words into one half-word, which emerges as the *scrambled* form of the name. There is a danger here that two totally different names might accidentally scramble into the same half-word, but this has not happened with the few hundred names of the existing vocabulary.

The scrambled form of the name is stored in a table of 1024 words. For quick access to this table, the bottom 9 bits of the scrambled word are used as an address in the table; if the table word so found already contains a scrambled name, the next is tried, and so on until there is a space to put the new scrambled name. When a word is being sought in the table its approximate position is then known, and it is not necessary to search the entire table to find it. This technique is only possible because there is much more space in the table than there are entries for it.

When the same name is read from the user's preface, it scrambles into exactly the same half-word. For the purpose of scrambling, all spaces and decimal points are ignored, and upper case and lower case letters are treated alike, so that "Lattice type" and "1 ATT ice TyPe." would produce the same scrambled form. The 1024-word table is then used to look up information about the name.

7. Data names

If the name refers to some item of data (a *data name*) the table would indicate this, and give the system the address of a piece of program which will read in the expected item of data. For example, "$a =$" would initiate the reading of one decimal number from the preface, to be stored in the part of the store where subsequent pieces of program would expect to find the length of the unit cell side conventionally called *a*. "Lattice =" initiates the reading of another name which in Fig. 2 was "C Face centred". This name would then be recognized as a plausible lattice type, and some identifying integer would be stored.

On a larger scale, "Atoms" initiates the reading of a *list* (in the non-computer sense) of atomic positions, each punched conventionally as

⟨label⟩ ⟨x coord.⟩ ⟨y coord.⟩ ⟨z coord.⟩.

As the number of positions varies from one crystal to another, this number must have been already read. The labels and co-ordinates are stored in preassigned places.

The Friend must know which items of data it has already read. A 4-word (192-bit) Boolean vector is kept, with one Boolean entry for each item of data which could possibly be read. This vector is referred to in three distinct places. Firstly during input of certain items of data: if, for instance, all six numbers which define the unit cell are present, a certain amount of arithmetic can be done which is required so frequently in calculations that it is sensible to do it once only and to store the results. It is possible to test whether these six numbers are present by consulting the relevant Booleans, and as soon as all six have appeared the necessary arithmetic is done.

Secondly, before an arithmetic routine is obeyed (see §8) the Friend must check that all data essential to the routine are present. In the example in Fig. 2, essential data are

a, b, c, alpha, beta, gamma, atoms

In practice we use an extra Boolean called "unit cell" to represent the six separate Booleans for a, b, c, alpha, beta and gamma. Together with the list of all possible names that is read first by the system, we have a list of *essential data*; the entry in this list for the Bond lengths routine would be

*Bond lengths	(the asterisk indicates an arithmetic
Unit cell	routine name rather than a data
Atoms	name)

On input of the vocabulary, this would be turned into a 4-word mask with a 1 in place for every item of data needed. The mask is then compared with that of given data described above, and the Friend refuses to try to obey an arithmetic routine if all its data are not present.

Thirdly, we have allowed many of our items of data to be optional. Much of the input to a conventional program is concerned with choices which the user does not necessarily wish to make, such as numbers to scale output, or maximum values of various quantities. Wherever the Friend can make some reasonable assumption if an item of data is not given it will do so. These assumptions must, of course, be made immediately before an arithmetic routine is obeyed, and in order to discover the presence of such items the Friend must consult its vector of given data. The apparent omissions in Fig. 2 as compared with Fig. 1 are all dealt with by the Bond lengths routine in this way; the absence of a maximum bond for which angles are to be calculated, for example, is assumed to imply that no angles are wanted.

8. Routine names

So long as we are reading and recognizing data names, we are simply assimilating data, with no idea of what is required subsequently. The other commonly occurring type of name is the *routine name*, such as "Bond lengths", "Least squares refinement" or "Stop". The routine name has no counterpart on the preface for a conventional program; the user would have selected a particular program by providing a paper-tape copy of it, or demanding it from a certain place on a magnetic tape. If he wishes to obey more than one program he has more than one run.

A routine corresponds roughly to one conventional program to do arithmetic. It can vary in size from a full-scale refinement routine of several thousand orders, to the single order "Stop" (which, of course, simply marks the end of the preface, and is a historical relic from the time when there was no easier way of doing so).

The routine name could be treated by the Friend in a straightforward way; on recognizing the name, the Friend checks that it has all essential data, reads down the routine from magnetic tape, and enters it. The routine first assigns reasonable values to any optional data which has not been given on the preface, and then performs the arithmetic requested, prints some results and returns to the system.

In practice, the user tends to think first of the routine he wants, and then of the data he must give it. Thus in Fig. 2, if we tried to obey "Bond lengths" as soon as it had been read from the preface we would have no data for it at all. So on recognizing a routine name, we remember this name, but do not actually call the routine until we find another routine name further down the preface. In this case, data would be absorbed until the name "Stop" was recognized, and only then would the Bond lengths routine be obeyed.

9. Sequence of programs

A routine returns control to the Friend when it has finished the arithmetic required of it. The Friend then continues reading the preface, obeying routines as they appear, in the delayed manner described above. In this way, a user can call several routines in sequence, or even the same routine several times. This feature is useful in crystallography for several reasons. For example, in the early stages of collecting observations, the user may require several different types of correction to be applied to them in sequence. In the past he has had to run several different programs to do this, at best preserving the partly-processed observations on magnetic tape, and at worst repunching them in different formats for every program. Again, when he is refining his suggested structure of a crystal, he may wish to try various values of one variable simply to see what happens. It would be quite plausible on a large and fast computer to use this apparently rather crude and wasteful technique on small amounts of data. A portion of preface reading

R factor
Number of new temperature factors = 1
Temperature factor = Na6 3·4

R factor
Temperature factor = Na6 3·6

where "*R* factor" is a routine which calculates how good a fit the user's suggested structure gives with his observations, would cause the Friend to perform the necessary repeated calculations of this fit, varying only the temperature factor of the atom labelled Na6 from one calculation to the next.

A sequence of programs also proves useful towards the end of a refinement, when it is sometimes the case that the computer can perform the refinement automatically. By this time all relevant data have been given, and such a preface would contain a list of the routines required and little else. For example:

Least squares refinement	*Comment*
No. of cycles = 4	(this could be omitted if the previous run had 4 also)
Difference density map	(where exactly what to print has become standard, and so is omitted)
Bonds	
Contour slant plane	(a routine using a curve plotter to give results to be published)

Equation of plane = $6 \cdot 2x + 3y + z = 4$

Stop

Such a sequence would replace at least four consecutive runs of discrete programs.

10. Synonyms

A system of data presentation based on words rather than numbers is liable to errors for various reasons. If a user has always called a standard routine "Distance-angle" he will probably forget that in the Friend it is "Bond lengths"; he may well find it difficult to spell "equi-inclination Weissenberg", or to remember whether or not the hyphen is present; if he is American he will want to refer to "reflexions" and "centered"; and even if he has written everything correctly, if a data tape becomes too verbose there is more chance of error in punching the tape.

We therefore introduce a system of synonyms. When the vocabulary available to the user is first read by the Friend it is listed in the form

⟨Name⟩ ⟨Number⟩

where the number gives an entry in a table linking the name with a suitable piece of program. Thus a typical entry might be

Maximum bond = 64

where entry 64 in a table points to a piece of program which reads a number and stores it in a place where a later routine will expect to find the maximum bond. By convention, if an item of the vocabulary is preceded by an asterisk it is expected to be a *synonym*, and to have as its ⟨number⟩ a number which has occurred before. It is an easy matter to treat the two names identically; the scrambled synonym is added to the 1024-word table (described in §6) pointing to the same piece of program as does the name for which it is a synonym. This facility takes no extra store, as the 1024-word table is only sparsely occupied, and it gives a very wide range of vocabulary. We can cater for the user in a hurry by making "EW" a synonym for "equi-inclination Weissenberg"; for the American by "reflexion" for "reflection" wherever it occurs, and even for languages other than English if the need were to arise.

The present vocabulary consists of about 500 names, and about 350 of these are synonyms. If we find a user making some plausible mistake in punching a name, we can add his version to the vocabulary as a synonym.

As the Friend may need to refer to items of the user's data by name, we keep a packed, character form, version of all names, but not, of course, of synonyms. Together with these packed names we store other words and sentences, to be used as headings for output or diagnostic messages. These messages and headings are read by the Friend in exactly the same way as are the names of the users' vocabulary.

11. Structure of the system

The largest part of the Friend is naturally the set of arithmetic routines. These are held ready assembled on magnetic tape, in a form suitable for transference to a fixed part of the store (*the routine space*) when they are called for by the user. Arithmetic routines together with their working space vary greatly in size, and any store not required by a particular routine is given back to the Supervisor. Thus the space used may expand or contract in order to make the most efficient use of the computer. Those routines of the Friend itself which are not constantly in use, such as input routines, are also held on magnetic tape and read down only when they are needed.

Below the routine space is the *user's data space*, into which all the data given on the preface are stored. This space must be accessible to every arithmetic routine. As amounts of data also vary greatly from one crystal to another, it would be desirable to allow this data space to expand and contract as does the routine space.

Below the user's data space, at the start of the store, there is all the program and working space which needs to be present permanently. The program here includes the small controlling program, the output routines and magnetic-tape routines.

12. Control program

The tools available to the input section of the control program are:

(*a*) a routine to read a name and scramble it,
(*b*) routines to put a name into the 1024-word table and to look up a name in it,
(*c*) various conventional input routines to read decimal numbers, integers, or data in special formats such as $(1/2+x, 1/2+y, z)$.

The number routines have been written specially, because it is useful to be able to read a number terminated by any symbol, as in

Alpha = 87degrees 42minutes

Input tapes for the Friend are usually produced by a Titan Flexowriter, which possesses a "back-space" and a "tab". The input is therefore "line-imaged"; that is, it is read a line at a time, and an image is stored corresponding to the print-out of the tape. Multiple characters are allowed, so, for instance, "*θ*" is a synonym for "theta".

13. Preliminary reading of vocabulary

The Friend is stored on magnetic tape already assembled and with all the vocabulary available to the user "built in". It would be tedious to build in the scrambled form of names, or even the packed messages suitable for output, so use is made of the routine (*a*) of §12 to read and store lists of various names before the preface is read. The lists which comprise the whole vocabulary are:

(*a*) All names expected as data names, and their numbers and synonyms.
(*b*) All names expected as program names, and their numbers and synonyms.
(*c*) For each program, a list of all the essential data.
(*d*) All error messages.
(*e*) All headings, or groups of symbols required to be output.

When a change is made to the program or the vocabulary, the program is reassembled, the vocabulary is read and stored, and the entire system is filed on magnetic tape.

14. Interpreting the preface

The preface is "interpreted" in the sense that it is not compiled; the instructions of the preface are obeyed as they appear.

The simplified flow diagram for the control program, disregarding all use of magnetic tape for data, which will be described in §15, is shown in **Fig. 3**.

The process stops when the routine which is about to be labelled "previous" is "Stop".

15. Magnetic tape

The technique of allowing several routines to be called sequentially in the same run allows data common to all routines to be given once only; what the user would really like to do is preserve his data and intermediate results from day to day, or even month to month. The only removable form of storage on Titan is a reel of magnetic tape.

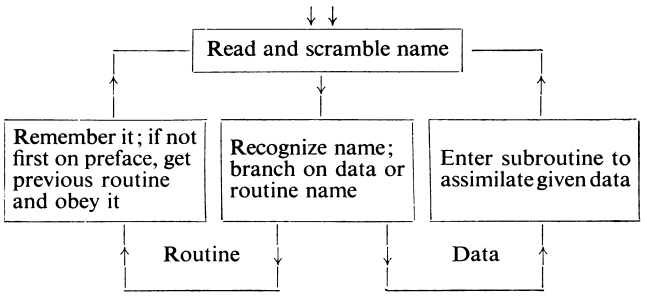

Fig. 3.—Simplified flow diagram of control program

Ideally, each user has his own magnetic tape; in practice this would be so operationally inconvenient that we must devise some system of using one long tape for, say, 10 or 20 users. At present the Friend is held on this same tape, but it may later become part of the magnetic disc filing system currently being developed on Titan.

In order that a crystal should be recognizable from one run to the next, its data are given a title (such as "Osmium Tetroxide" in the example in §5) which is read as a "name" and scrambled exactly like any other name. If the first name of a preface is not a recognizable data or routine name, it is assumed to be the name of a crystal. From the magnetic tape a dictionary is read down and scanned for the scrambled form of the crystal name. Assuming the user has been doing calculations for this crystal for some time, there may be several dictionary entries for it; the *first* of these points to blocks on the tape which hold the latest copy of all the data ever read for this crystal. These data are read down into the store into the user's data space of §11. Any new data read from the current preface will overwrite this, so the user has the option of keeping any particular item or reading in a new value for it. The 4-word Boolean vector describing which items have been read is also preserved from run to run. At the end of a run the current state of the user's data space is written back to the magnetic tape on top of the copy which was read at the start.

Users also wish to store intermediate results; one common example is in the collection of observations from an automatic diffractometer. Observations may be being measured at a rate of, say, several hundred a day, and the user would like the first set to be examined for plausibility before subsequent sets are measured. The observations, then, are presented for processing in distinct sets, to be concatenated finally. Each set is partially processed and then written as a *document* to magnetic tape and given an entry in the dictionary.

Various types of document are stored in this way; when a set of suggested atomic positions is improved by a refinement routine into a better set, the original positions are put as a document on to tape in case the

user wants to revert to them; several sets of observations (depending, perhaps, on whether or not certain dubious ones are included) could also be held, each set having been given a serial number for reference. In fact, new types of document occur whenever a routine is added to the system.

16. Error detection and information for the user

If a system is going to perform calculations for a user making assumptions about the user's data, either because the user has never supplied the data and wants some reasonable values to be assumed, or because he supplied them weeks ago and does not want to do so again, the user must know what assumptions are being made. A balance must be struck here between withholding information the user would consider essential, and printing out everything remotely connected with a calculation every time it is performed. The latter will in some cases produce so much printing that the user will ignore all of it.

The general principles which have been followed are:

(*a*) always to point out any oddities,
(*b*) in general to suppress printing of more usual information, but to have it available if the user requests it.

(*a*) includes the detection of all sorts of error. If, for example, a set of "space group operators" does not form a group (in the mathematical sense), it is pointless to proceed with any calculation. The Friend's suspicions of what is wrong are printed out in English, together with helpful information such as the last line read from the preface (in case the trouble stems from a tape-reader error) and any suggestions the Friend might have for remedying the situation. In this particular case the Friend would stop altogether, as the error is too serious to correct. On the other hand, if a user gives a strange value for some physical quantity this would be pointed out to him as disturbing, but the calculation he had requested would proceed, on the grounds that he has probably some good reason for it. In the same category would come the Friend's reaction to being told "$c = 5{\cdot}3214$" (where c is the length of a cell side, and unlikely to change once measured) after it has been working with $c = 5{\cdot}2314$. The user would be informed of the change and the new value would be accepted.

It is considered essential to take a great deal of care in giving all diagnostic information. When a user is almost entirely separate from the computer, he must be protected from all forms of machine-dependent error indications. For instance, in machine code programs when anything goes wrong, Titan will print a "monitor" which is helpful to the programmer but unintelligible to the user. All possible monitors, therefore, are trapped and turned into messages like:

"Square root of negative number; should not happen; program or machine error: try again and see Judy if it happens twice".

If a user wishes to have certain features pointed out on his output he can take advantage of the "comment" facility. As described in §10, if a name starts with an asterisk this fact is detected specially, during the reading of synonyms in the vocabulary. This feature is used during the reading of the preface in a different way. A line of preface preceded by an asterisk is treated as comment, and copied straight to the output and ignored. In this way the user can, if he wishes, describe in detail any of the features of his run which might not otherwise be apparent when the output is studied later.

Principle (*b*) demands a number of decisions. In order to deal with these, we have various Boolean items of data which are relevant to one run only, and are *not* preserved from run to run. Almost all of these are indications of whether or not to print any particular quantity which is a by-product of the calculation. For example, the routine which produces Lorentz and polarization corrections calculates for these the quantities "theta" and "s squared" which the user may or may not want. The user's vocabulary contains the names

Print theta
Print s squared

which set these Booleans for this particular run. This general scheme extends to all "optional" output. If any such output is not asked for explicitly it is not given.

17. Information service

Following ideas from Project MAC at MIT (see, for example, Neisser, 1964), as much information about the system as possible is held with it on magnetic tape. This includes both information for users (up-to-date copies of the users' manual, the date when the manual was last changed and what the changes were, sample prefaces for standard calculations, and anything else considered to be remotely helpful to the user) and for programmers (lists of allocation of working space, the theory behind the arithmetic routines, detailed specification of every routine and so on). All this information is accessible in exactly the same say as is the Friend itself. A preface saying, for example,

Print changes to users' manual
Date of my manual = 11.3.66

would cause to be obeyed a routine which examines the dates of changes to the manual and prints any occurring after 11.3.66.

Keeping detailed information about the program itself is essential for a large machine-code system. This information is used and updated whenever the Friend is changed, and it is to be hoped that it will be useful to the next programmer who is called upon to maintain the system.

18. Record of runs for a crystal

On each magnetic tape is held a record, or log, of all runs involving the tape. The log includes the name of the crystal, date run, time taken and an indication of what

happened in that run. A user can then, if he wishes, trace the past history of his job, either out of curiosity or if he is in doubt over any particular run. The programmer can also keep track of how the system is being used, and how long a magnetic tape lasts.

19. Future development

The Friend as described was developed under the Temporary Supervisor of Titan, and became obsolete when the Main Supervisor came into service. Titan now has an 8 million-word magnetic-disc file, and when the Friend is rewritten it is intended to use some of this to hold everything except the data and results which the user actually wants to remove from the computer. Thus a casual user with no permanent data on magnetic tape should be able to supply for a run only his preface and a "Job description" which can be as short as

Cryst
[Name of job]
Terminator

It is anticipated that it will be increasingly easy to add arithmetic routines to the system, for two reasons. Firstly, input of its data has now been separated from an arithmetic routine; the routine is written assuming that all its data is present. For a new routine only those items of data which have not occurred before need to be catered for, and even these usually fall into categories such as "read one decimal number". Secondly, many subroutines useful to a new program will be found to exist already and be known to work.

Development of the Friend so far has been an effort to keep up with the demand for standard crystallographic programs in a fast-changing world. Once we have a workable system which can be left to deal with everyday requirements, we can build on the system, and, for example, experiment with new methods of refinement, and deal with the control of new automatic apparatus as it appears.

20. Acknowledgements

The author is grateful to Professor Sir Nevill Mott and Dr. W. H. Taylor of the Cavendish Laboratory for their encouragement and for the provision of facilities, and to Professor M. V. Wilkes and the staff of the Mathematical Laboratory for allowing and assisting her to use the Titan computing service. The author also acknowledges with gratitude financial support provided by an S.R.C. grant.

21. References

BARRON, D. W., BUXTON, J. N., HARTLEY, D. F., NIXON, E., and STRACHEY, C. (1963). "The main features of CPL", *The Computer Journal*, Vol. 6, p. 134.

HEXT, J. B. (1965). "Programming languages and Compiling techniques", Ph.D. Dissertation, Cambridge University Mathematical Laboratory.

NEISSER, U. (1964). "MAC and its users", M.I.T. Memo. MAC–M–185.

Book Review

Handbook for Computing Elementary Functions, by L. A. Lyusternik, O. A. Chervonenkis and A. R. Yanpol'skii, 1966; 251 pages. (Oxford: *Pergamon Press*, 63s.)

Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables, edited by M. Abramowitz and I. A. Stegun, 1965; 1046 pages. (New York: *Dover Publications Inc.*, 32s.)

The titles of these two books indicate that their intentions are not the same. Since the Soviet book limits itself to elementary functions one might expect its coverage of these to be more comprehensive, and sometimes this is so. Coefficients of approximation formulae for these functions are often given to suit a wider range of requirements of accuracy than in the American book. Occasionally one finds surprisingly more detail about a higher function in the first book, for example the Gudermannian Function discussed in an appendix. It is interesting to have details of algorithms used on Soviet computers but the descriptions waste space since the algorithms used with two machines are often the same.

In spite of the few reservations made above, the second book surely represents much better value for money. It is four times bigger, just by page numbers, and half the price. The tables are more elegantly displayed and easier to use with the five-digit separation. Graphical display of some functions is very useful, entirely missing in the first book. The collected formulae at the beginning of the sections are probably the greatest value of the book, but in making these comparisons one is again reminded of the titles of the books.

Where direct comparison of numerical values can be made, a number of checks fail to discover any discrepancies between them. This says a great deal for the care with which the books have been prepared. They are both valuable but I guess that the paperback construction of the Dover book will be severely tested by constant use, in spite of the claims made (on the back cover) for its durability.

R. J. ORD-SMITH