Fundamental principles of expressing a procedure for a computer application

By T. R. Thompson

An examination is made of the fundamental needs of a language to be used for expressing procedures for all types of computer applications. An outline is given of the commands needed for doing calculations, controlling the sequence of steps, obtaining the information required, and presenting the results in the desired format. Further, the particulars that must be provided about data and results are indicated. The main features of the CLEO language are described in an Appendix.

1. Purpose of paper

It is sometimes contended that the main reason for developing programming languages is to enable a person not versed in computer techniques to set down the procedure he wants carried out in such a way that this can automatically be translated into a computer program. The professional training of the person who may want to do this varies enormously; he may be a scientist, a mathematician, a statistician, an actuary, an accountant, or an office manager. For some reason it has always been assumed that the only language an office manager could use is ordinary English. However this may be, one would certainly suppose that the basic need is for a simple language. Yet the published statements on most programming languages are very difficult to follow, even by people with long experience of computers; perhaps this arises from an attempt to avoid possible ambiguities which is, of course, important because the language must be precise. The aim of this paper is to express in simple terms what seem to be the fundamentals of a language which is to express the procedure to be carried out for all types of applications. It seems clear that, if the attempts in Europe and elsewhere to devise a common language for computer users are to be successful, the fundamentals must be agreed first, and each feature agreed then considered in turn to decide the best way of giving effect to it. At the same time the way must be left open to expand the language as experience reveals the need for further facilities.

It will be seen, therefore, that this paper is solely concerned with devising a suitable language, not with the preparation of the compiler, though naturally the language devised must be one that lends itself to translation by a compiler.

2. Basic types of commands to carry out procedures

Any procedure consists of a series of defined steps which may conveniently be referred to as *commands*. These commands are of three main types.

(*a*) Those required to calculate one value in terms of others.
(*b*) Those required to control the sequence of steps in the procedure according to circumstances.
(*c*) Those required to obtain information provided as data and to present results in the desired format.

3. Information used in procedures

Since almost all commands are concerned with information of one kind or another, it is important that the language should provide a means of identifying and describing information as well as giving commands. The three main types of information to be used are as follows.

(*a*) *Data* which is provided for the purpose of carrying out the procedure, including fixed data such as constants and tables of information.
(*b*) *Results* which are to be formed by the procedure and recorded for the use of people.
(*c*) *Intermediate results*, i.e. those which are formed by the procedure solely for use at later stages of the computation.

It is convenient to regard any associated pieces of information as a *file* containing a series of *records*, whether these records are on punched paper tape, punched cards, magnetic tape, a series of printed forms, or in compartments of a computer store.

The plan for the application must give a complete description of the data and results (but not necessarily of intermediate results). The language, to be complete, must provide formal means for describing the file, and the description for each file must include the following.

(i) The layout of the file must be given, breaking it down to records, subrecords, groups of items, and individual items of information; that is of all levels in which the information is arranged.
(ii) Each item, group, record, etc., must be identified by name, number, or other symbol as convenient to the circumstances.
(iii) The form in which each item of information is to be represented must be specified precisely; for instance, whether it is to be alphabetic, numeric, or a combination of them, and if numeric, in what unit and notation the number is to be expressed. Usually the maximum size will need to be specified, and for result numbers in particular it will be necessary to specify the layout and punctuation, the decimal point, blank spaces between digits, suppressed zeros, and a place for the + or − sign.

(iv) The medium on which the information is to be presented must be given and, if an item is to be identified by its location on a punched card or printed statement, the line and position on the line in which the item is to appear.
(v) Where several entries may appear for any item, any group of items, or any record, a means will be needed of showing the number of entries which will appear or, alternatively, a maximum number.
(vi) Where alternative entries may appear, e.g. a receipt or an issue of goods, there must be a means of showing this.

The extent to which data descriptions need to be explicit or implicit is a matter for discussion in the light of experience. It may be, for instance, that "sterling" can be taken to imply a definite layout and punctuation arrangement and, provided this will suffice, no further description is necessary. Essentially the extent of this formal description of the information must be such that, once the procedure has been written purely in terms of the commands and the identifiers of the pieces of information on which the commands are to be applied, a compiler can, by reference to the description, produce a suitable computer program. Thus if consistency checks on the data are required to be put in the program by the compiler, the data description must indicate the nature of consistency to be expected.

4. Commands for doing calculations

As indicated in the descriptions given of several languages, only one command is needed to calculate the value of any item from the value of others. The command will, in general, incorporate a formula specifying the calculations. How the formula is expressed on paper may be varied, but the essential is to link a series of operands by appropriate operators. The range of operators required depends on the variety of applications to be covered.

The simple arithmetic operators are essential, together with the means of getting a remainder from the division process. Mathematical operators, such as sine, log, may be needed, as well as scaling, floating-point operators, etc. If actuarial applications are to be considered, there may be need for an operator to calculate "amounts" in terms of "principle" and "interest" according to specific formulae.

Rounding off, up, or down, are likely to be needed, and it is convenient to be able to include a conditional expression in a formula.

Essentially therefore the single command to "calculate" needs the facility to include a variety of operators, which may be added to as experience shows this to be necessary.

5. Sequence of steps in procedures

Commands are needed

(*a*) to control jumps, conditional or otherwise;
(*b*) to determine repetitions of one or more steps.

The jump command must indicate to which next step the jump must be made and under what conditions. Where the jump is conditional it is convenient to preface the command by the condition, expressed in terms of equality or inequality of values or expressions. For this purpose an expression may be set down as a formula in the same way as for doing calculations.

There must be provision for multiple and for alternative conditions, and also for combinations of them. Another need is to be able to jump to one point in the sequence of steps if a condition is satisfied and to another point if not. This need can be extended to provide for a "switch" which permits jumps to one or other of a series of points according to different possibilities.

For repetitions the command must indicate the step or steps to be repeated and either

(i) how many times they are to be repeated;
(ii) until some condition is satisfied;
(iii) for values of some "identifier" over a given range.

6. Obtaining data and presenting results

Commands are required that will get data

(*a*) from the next record from a file;
(*b*) from the next record of a specified type;
(*c*) from a particular record, i.e. one with a given key number or name.

A command is also required to put away a given record in a file.

Other useful facilities in dealing with files are

(i) to copy from a data file to a result file;
(ii) to sort or merge information on files;
(iii) to extract a particular item of information from each of the records of a file in turn, as, for instance, the ledger balances of customers' accounts to check the control total.

7. Features and configuration of a particular computer

It will be noticed that, in outlining the fundamentals of a language for expressing a procedure, no reference has been made to the features of the particular computer which is to do the job. Clearly it is desirable that the language shall be capable of expressing the procedure whichever computer is to do it. On the other hand, the plan for the procedure must have regard to these features: the types of peripheral equipment, the capacity of the store, and whether an auxiliary store is attached or not. The effectiveness of the plan will clearly depend on making the best use of these features. It is, however, in the compiling stage that these features must be taken into account. The compiler will of necessity be designed to cope with types of computer within a given range of store capacity and peripheral equipment, and when translating, the parameters for the particular computer must be fed in before the program which is to be translated, so that the resulting object program is one that can most conveniently do the job on this computer.

8. Summary of principles to be observed in devising the language

The language must

(*a*) be one which is simple enough for people to learn and use quickly and easily;
(*b*) be as concise as possible, but use no symbolism which is not convenient for people without special training;
(*c*) use as few words as possible in a special sense;
(*d*) provide adequate means for describing in a natural way the different aspects of procedures and associated information;
(*e*) be completely precise;
(*f*) avoid any complications that are not absolutely essential to providing the required facilities, so as to keep the compiler as simple as possible.

9. CLEO

CLEO was devised in accordance with these principles, which were set down after a study of COBOL and other languages in the light of our own experience.

As requested at the British Computer Society Northampton College meeting at which this paper was first presented, a summary of the main features of CLEO are given in the following Appendix. It assumes an understanding of the technical terms normally used in connection with this subject. A word (or abbreviation) in block capitals indicates that this word is reserved for the special sense in which it is used literally in the language. On the other hand, words in italics indicate the nature of the expression, word, number or symbol to be used. In writing the language the actual expression, words, etc., appropriate to the application would be substituted.

Appendix

The CLEO Language

1. The types of Command.
2. Data and Result Descriptions.
3. Overlaying of Information.

1. Types of command

Commands may be either imperative or conditional. Conditional commands are executed or not according to some condition incorporated in the command after the words "IF" or "WHEN." In what follows the word "Command" is used without qualification to mean a command that is either imperative or conditional.

1.1. *Commands for doing calculations*

SET followed by

"*Identifier of the result* = *Arithmetic expression*"

NOTE. The arithmetic expression consists of operators and operands (or functions of operands) combined together according to the rules of algebra. Matching brackets may be inserted to group terms together.
The operands may be literals or identifiers of data or intermediate results. The functions may be exp, log, sin, cos, etc. The operators may be addition, subtraction, multiplication, division, exponentiation. After division, rounding off or up may be specified by "(RO)" or "(RU)" after "SET." The remainder after division may be obtained by an additional clause, following the SET command, of the form

"AND *Identifier* = REMAINDER"

1.2. *Commands for controlling the sequence of steps in procedure*

GO TO followed by

"*Procedure label*"

for the step to be carried out next,

or, to denote a switch, by

"$S : P_1, P_2$, etc."

where S is an identifier which takes integer values 1 to N and where P_n denotes the procedure label for the step to be carried out next when the switch is set at n.

NOTE. The procedure label may be numeric or alphanumeric.

DO followed by

"*Subroutine label*"

to execute a subroutine.

NOTE. The subroutine label may be numeric or alphanumeric. After the subroutine has been executed, the next step is normally to execute the command following the DO command, but a GO TO command may be obeyed within the subroutine leading to some other step outside the subroutine.
In carrying out the subroutine, however, the same subroutine may not be re-entered either directly or indirectly until the execution of the subroutine has been completed.

HALT followed by

"PROGRAM"

to terminate the program

or, where it is desired to allow the console operator to choose one among several possible continuations from a particular point in the program, by

"*I*"

immediately followed by another command

"GO TO P_1, P_2, etc."

where I is a reference number which is displayed to the console operator on an indicator to indicate the nature of the stoppage. The operator may then choose to continue the program with any one of the procedures $P_1, P_2, \ldots, P_n, \ldots, P_N$ when n is the integer set up by the operator on the controls.

IF followed by

"*Conditional expression Imperative command*"

or, by

"*Conditional expression Imperative cor. 1*

ELSE *Command*"

where some special course of action is desired if the condition is not fulfilled.

NOTE. A conditional expression consists of a relationship or several relationships joined in pairs by the words "AND" or "OR." "AND" takes precedence over "OR," but where "OR" is to take precedence brackets are needed. The relationship may consist of two arithmetical expressions separated by "$<$," "$\leqslant$," "$=$," "$>$," "$\geqslant$," "$\neq$." In particular one or other or both of the arithmetic expressions may be simply an identifier, and one could be a literal.
Other conditions may relate to identifiers of records of input files (see under OBTAIN)
Another form of conditional expression is

"END *File identifier*"

to deal with the case when the end record of a given file has been reached.

FOR followed by

"*Identifier* = *Control expression Imperative command*"

to cause repeated execution of an imperative command.

The control expression may take one or other of the following alternative forms:

(i) "A_1, A_2, A_3, etc." to give a series of values for the identifier determined by these arithmetic expressions
(ii) "$A_1 : A_2 : A_3$" for values from A_1 by steps of A_2 to A_3
(iii) "$A_1 : A_2$ UNTIL *Conditional expression*" for values from A_1 by steps of A_2 until the condition is satisfied
(iv) "A_1 UNTIL *Conditional expression*" for values of A_1 until the condition is satisfied.

Any succession of control expressions are permissible consisting of

(*a*) repetitions of one form

and/or (*b*) different forms.

NOTE. The command to be carried out may in fact be a series of commands preceded by

"BEGIN" and terminated by "END."

WHEN Used in conjunction with "SKIP" and "COPY" (see later).

1.3. *Commands for obtaining data and presenting results*

OPEN INPUT followed by

"*Input file identifiers*"

to enable the procedure for checking the file headings to be provided by the compiler.

OPEN OUTPUT followed by

"*Output file identifiers*"

to enable the procedure for loading the files to be provided by the compiler.

CLOSE INPUT
or
OUTPUT followed by

"*File identifier*"

to enable the procedure for closing the files to be provided by the compiler.
For a magnetic-tape file "NO REWIND" may follow a file identifier to indicate that the tape is not automatically to be rewound.

OBTAIN followed by

"*Input file identifier*"

which is used to obtain the next record from a file which is set out in some standard format, as on magnetic-tape files,

or, when it is desired to obtain the next record only if it is of a specific type, by

"*Record identifier*"

If the next record is not of the specific type no record is obtained.

NOTE. In either case, if action is required only when the record is of a given type, a conditional command may follow with a condition as follows.

"IF *Record identifier*"

READ followed by

"*File identifier*"

which is used to obtain the next record from

a file where the format varies according to the application, such as on paper tape or punched cards,

or, when it is known the next record is of a specific type, by

"*Record identifier*"

which ensures the record is treated as being of this type.

NOTE. In the former case a subsequent command may be made conditional on the record being of the type required.

SKIP followed by

"*File identifier*

WHEN *Conditional expression Command*"

This enables types of records on which no action is required to be skipped, and appropriate action to be taken on those where it is required.

The conditional expression may be of any form, multiple or compound, including

"WHEN *Record identifier*"

to ensure appropriate action when the record obtained is of the type shown,

"WHEN *Identifier* = *I*"

where *I* is a literal

to ensure appropriate action when a specific record is reached.

A list of "WHEN" clauses may also be used.

Any type of command may be used in the "WHEN" clause, including a further "SKIP" in order to find a lower-level identifier.

NOTE. Where there are more "WHEN" clauses than one, then it is understood that when any condition reached in the list is satisfied, the command corresponding to this is executed irrespective of later conditions that might be satisfied.

COPY followed by

"*File identifier* TO *File identifier*"

Again as in the case of "SKIP" this may be followed by conditional clauses to arrange an alternative course of action when one or more conditions are satisfied.

NOTE. The same arrangement of multiple "WHEN" clauses applies.

FILE followed by

"*Output file identifier*:

List of output identifiers FROM *List of input and internal identifiers*"

which is used to assemble and record information on a file which is set out in some standard format, as for instance on magnetic-tape files.

There may in fact be more than one pair of lists in which case the second and subsequent pairs are each prefaced by "AND."

In order to cope with the case where an indeterminate number of items has to be filed, the command may be limited by a clause

"UNTIL *Conditional expression*"

where the condition indicates that no further items are to be filed when the condition is satisfied, e.g. if the end marker of a list of items has been reached.

If it is desired that the final item when the condition is satisfied is also to be filed, the clause

"UNTIL *Conditional expression*
INCLUSIVE"

is used.

NOTE. The lists of identifiers may be of items, groups of items, records, etc. One identifier on the output list may be related either to one on the input or internal list or to several of these.

WRITE followed by

"*Identifier of output file*:

List of identifiers FROM *List of identifiers*"

which is used to assemble and record information on a file where the format varies according to the application, such as on paper tape or punched cards. The arrangements for this are exactly as for FILE.

PRINT followed by

"*Identifier of printer file*:

List of identifiers FROM *List of identifiers*"

which is used to assemble and print information.

The arrangements for this are the same as for FILE except that a paper throw may be indicated by the insertion of " : " after the last item to be assembled for any line.

MOVE followed by

"*List of identifiers* TO *List of identifiers*"

which is used to move information from one set of input or internal items to an internal set.

The same arrangements apply as for FILE.

CLEAR followed by

"*Identifier.*"

This is used in order to assign zero to numeric items and blank to alphabetical items of the identifier.

2. Data and result descriptions

2.1. *File heading sheets*

For each file the programmer completes a heading sheet to give basic information about the file.

(*a*) *For magnetic-tape files (and for internal records)*

(i) Whether input or output
(ii) Block size
(iii) List of records in the file
(iv) List of control totals to be kept for reconciliation purposes.

(*b*) *For a paper-tape file*

The same as for magnetic tape and, in addition, whether there is to be a fixed or variable field format.

(*c*) *For a punched-card file*

The same as for paper tape except that the block size is not required.

(*d*) *For a line printer*

(i) list of records in the file;
(ii) number of lines in the record;
(iii) type of printer to be used.

2.2. *File description sheets*

For each file there is a description sheet that sets out in columns for each identifier to be found in the file particulars appropriate to that identifier. These particulars for magnetic tape, paper tape, punched cards and line printer are given below.

(*a*) *For magnetic-tape file (and for internal records)*

(i) the "identifier" by which items and groups of items are referred to in the commands;
(ii) an integer representing the "level" down to which the file must be broken to read this identifier; the main subdivisions of the file have the level 1 and as the information is broken down to smaller pieces the level numbers increase until the "items" that relate to the lowest entity of information are reached;
(iii) extent of a record, to show how much information is to be obtained in response to an OBTAIN command. "*R*" is put in this column for the highest level identifier of the record, so that all information that follows is obtained up to the point when an identifier of equal or higher level is reached;
(iv) occurrence, to indicate the actual or maximum number of times this item, or group of items, may occur;
(v) representation, to indicate the manner in which an item of information is expressed: numeric, alpha or alphanumeric; decimal, sterling or binary; floating point;
(vi) position of integer point for each numeric item;
(vii) size of each item, indicating the actual or maximum number of digits or characters;
(viii) any rounding required;
(ix) whether any group of items is identical in description with some other group already described.

(*b*) *For a paper-tape file*

The same as for magnetic tape and, in addition:

(i) radix pattern for each numeric item in a notation other than decimal, sterling, or binary;
(ii) whether an item may be signed, and the form of representation of sign if this is special;
(iii) criteria by which particular types of record may be recognized;
(iv) if the information is to be in fixed field form and is to be printed off-line, e.g. on a teleprinter, the editing information as under (v) and (vi) for a line printer.

(*c*) *For a punched-card file*

As for paper-tape file and, in addition:

(i) column position on card of last character of each item;
(ii) spacing between repeated items or groups.

(*d*) *For a line printer*

As for magnetic tape but, in addition:

(i) radix pattern, as for paper tape;
(ii) indication of signed items, as for paper tape;
(iii) column position on paper of last character of each item;
(iv) line position for each group of items that constitute a line, or spacing from previous line;
(v) whether non-significant zeros are to be suppressed;
(vi) special editing arrangements, e.g. insertion of integer points, £, s, d, and other symbols, spaces, etc.

3. Overlaying of information

To provide for cases where the job does not require particular steps in the procedure or pieces of information after given points in the procedure have been reached, an Overlay Statement is prepared to provide the means for overlaying in the computer store. This statement also details pieces of information which are alternates to each other in that one and only one of the pieces may be present at any time. On the Overlay Statement are shown:

(i) details of those steps and records which are grouped into segments which may overlay each other, together with the point in the procedure at which the overlaying may take place;
(ii) identifiers of the sets of items, of groups, and of records which are alternates to each other.