A B S T R A C T

The increasing use of computers for data processing and the storing of large text files on magnetic tape and disk require methods of amending these files. The type of editing program needed depends on whether the file is of "flexible" text, as in the research being carried out at the Stanford Research Institute and at Brown University, or of the paragraph-based text of most books, with the format of the printed output of first importance, or of the comparatively simple "line-oriented" text of, for instance, computer programs. Editing programs are designed to make arbitrary alterations, or very simple repetitive ones, rather than the systematic changes accomplished by macroprocessors and string-manipulation languages. The construction of an editor is influenced by such factors as whether it is to be available for various purposes at various installations or for specific use in one, and whether the editing is to be on-line (in which case the type of console is relevant) or background. These and other factors in editor design can be illustrated by an examination of the interesting features of certain editing programs.

The editing program that is the particular subject of this paper is designed to modify "line-oriented" text. While implemented on an IBM 360 computer under the RAX and 44MFT operating systems and primarily for interactive use at the 2260 display terminal, it is intended to be machine independent and useful for background as well as on-line editing. In addition to the basic editing operations of insertion, deletion and replacement, both of complete lines and of characters within a line, the program includes facilities for interchanging blocks of lines in a file, for performing simple repetitive operations, for recovery from errors during editing, for checking the progress of editing by the printing of specified information, for handling files of different line lengths and for changing input/output devices. The editor uses either line number or context to specify points in a file and creates a new file during editing. The most-used editing commands are as brief as consistency and mnemonic value allow. Fairly detailed error and information messages are included.

The following thesis is my own composition, based on research carried out by me and no part of it has previously been presented in application for a higher degree.

JEAN BLUE

C O N T E N T S

PREFACE

The editing program described in the following pages was written in 1971 for use in the Computing Laboratory of the University of St Andrews with the operating systems, 44PS and RAX, of the IBM 360/44.

There was no previous means of editing disk and tape files off-line. Under RAX (Remote Access Computing System) limited on-line editing facilities were available.

The ASR33 teletype had an activity mode (/UPDATE) in which insertion and deletion of complete lines was possible. Line numbers had to be specified. These could be found or confirmed by a command (/DISPLAY) which printed at the terminal either the whole file or a particular sequence of lines.

The 2260 display terminal also had an UPDATE mode in which files could be amended. The method involved displaying the file on the scope, eight lines at a time, each with its number - a multiple of ten, to allow for the insertion of lines. Small changes were made directly to the text on the screen and a special character was typed against any altered line. A line was deleted very simply, by typing a special character in column 1. Insertions were made by typing the extra lines on lines 9 to 12 of the scope and indicating, by assigning sequence numbers to them, their position in the file.

For short files or for files where no large sections were to remain unaltered during editing, this UPDATE provision was simple and adequate. In other circumstances, it had three main defects. One was that the user had to "turn" each eight-line "page" of a file by typing the instruction "/PAGE". Even if the first ninety-nine lines of a document needed no alteration, the hundredth line (on the thirteenth page) could not be reached without the display of the previous twelve pages. Another was that the method of line deletion, although simple, became tedious if many successive lines had to be deleted. The third disadvantage was that if many successive lines had to be inserted, either several editing passes were made or several original line numbers altered (since only nine numbers were available between any two lines of the original file). Whichever way was chosen, a long insertion was awkward to perform, since only four lines could be added to each "page" on the scope.

Because of these limitations of the RAX UPDATE facilities, the present editing program was designed to be used on-line as well as to supply the want of a background editor for both RAX and 44PS. Since RAX has a conversational read feature, the editor may be used interactively - indeed that is assumed to be its normal on-line mode. It is not part of the RAX system, so cannot take full advantage of the 2260 CRT display capabilities and allow the user to indicate changes in the text by "pointing" with the cursor. It is possible to display on the scope for

inspection twelve lines of the file being edited, but not to have them read back by the system with amendments made to them. Instead, these amendments have to be expressed in words - or in code. This means that the same editing instructions are used on-line with a teletype or in a background job, as on the 2260. The editor allows the use of either line numbers or context as a means of specifying points in a file.

Since 1971, the 2260 UPDATE mode has been improved to enable the user to specify a line number or a required character string in order to access a particular position in a file, without necessarily displaying all preceding sections. Two context editors have become available, one [1] for on-line use only and the other [2] for on-line use under RAX and background use with 44MFT (which has superseded 44PS).

Before a detailed account of the present editor is given in Chapters 5 and 6, an attempt is made in Chapter 1 to discuss the need for editing programs at the current stage in the development of computer handling of text data files and what editing may mean in different contexts. In Chapter 2, the relationship of editors to two major means of manipulating text - macroprocessors and string-processing languages - is considered. Chapter 3 is concerned with the factors affecting the design of editing programs and Chapter 4 with some of the more unusual features of a few interesting editors.

I am indebted to Professor A.J. Cole, not only for suggesting the need for this editor, but for allowing me the privilege of entering his Laboratory - unfamiliar and exciting country - to learn at least a little of its languages and practices, to Mr A.J.T. Davie, my supervisor, for his help and patience during the spasmodic production of this essay, to Mr B.T. Mitchell, who must have wearied of hearing of my encounters with his temperamental RAX, to Mr J.M. Wilson and Mr R. Morrison, whose enthusiasm for the COTAN Amender was stimulating, although not infectious, and to staff of other Computing Laboratories, too numerous to mention individually, who supplied me - sometimes at great trouble to themselves - with information about editing facilities at their installations.
My thanks are due, too, to Mrs M.P. Sanders and Mrs C.J. Evans-Smith for their typing skill and forbearance.

CHAPTER 1

MACHINE-READABLE TEXT AND ITS MODIFICATION

This chapter deals with the need for editing - and with different interpretations of the term in different circumstances - where large text files exist.

1.a Provisional definition of editing

The definition of editing as "that process whereby text [of printable characters] held in some machine readable form is modified" [3] will be provisionally adopted, until the use of the term in the present context can be adequately illustrated. The rearranging of punched cards in a deck and the cutting and splicing of punched paper-tape are forms of editing and may even, with small files, be the most convenient ones.

In the last seven or eight years, however, the increasing provision of file storage space on magnetic tape and disk has brought with it the need for a simple way of modifying such files. They may well have been created on-line with no primary card or paper-tape input to be altered. Some form of editing program, therefore, is essential for on-line systems and desirable for any which support large, stored files.

Comparatively little has so far been published about editing programs, or editors, although it is clear that, with the increasing use of computers for data processing and the consequent creation of extensive text files, many such programs must have been developed. Some interesting research in the field has been done by van Dam and Rice, [4,5] but otherwise there are few published accounts. It seems from enquiry that most installations provide at least one editing program - often supplied commercially by the system manufacturer, such as the EDIT program of the ICL 4100 series, [6] or the #XMED and #XKYA of its 1900 series [7].

In order to see how both the idea and the practice of editing can vary in different circumstances and to define more precisely our particular field of interest, it will be useful to look at two related and extremely interesting fields in which large files of machine-readable text are maintained - that of research in text manipulation and that of printing and publishing.

1.b Research in text manipulation

The object of such research, championed by van Dam and Rice in the papers already mentioned, is to reach the point where text is thought of as flexible - capable of being presented and used in different ways, rather than in the fixed form to which we are accustomed on the printed, typed or written page. The process of creating, amending and using text on-line, at CRT display terminals, would go far beyond the already well-established practice of handling such files as computer programs in this way.

1.b.1 Augmented Human Intellect Research Center

A paper by Engelbart and English [8] describes the aims of the Augmented Human Intellect Research Center at Stanford Research Institute (AHI). The research is concerned with "changes ... in ways of conceptualizing, visualizing, and organizing working material". The authors state, "... we are concentrating fully upon reaching the point where we can do all of our work on line - placing in computer store all of our specifications, plans, designs, programs, documentation, reports, memos, bibliography and reference notes, etc., and doing all of our scratch work, planning, designing, debugging, etc., and a good deal of our intercommunication, via the consoles ... We are trying to maximize the coverage of our documentation, using it as a dynamic and plastic structure that we continually develop and alter ...".

Accepting that "one's concepts exist in a 'network' of relationships as opposed to the essentially linear form of actual printed records", the AHI data structure, used both internally and externally, is hierarchical - that of the tree. The user breaks his text into arbitrary segments called "statements", each one a node of the tree and bearing a number (e.g. 3b5a, 4a1 - the system adopted with slight modification in this paper) which indicates its serial position in the text and its level on the tree. Each statement may also be given a name, and named markers (externally invisible) may be placed anywhere in the text. This enables the user to examine his file in many ways, for instance by specifying particular nodes or certain levels of the file structure, by calling for the first line of each statement at one level in order to obtain a general outline of the document, or by specifying special markers.

1.b.2 Hypertext Editing System

One of the main objects of the Hypertext Editing System, (HES), [4,5] developed at Brown University, is to study, "applications of hypertext as a new medium". Van Dam and Rice and their colleagues have adopted the name, hypertext, and its meaning from Theodor Nelson. The user at his console organises his text into "areas". "Each area is a continuous linear string of text, like an Egyptian scroll, and might be a chapter, an entire book, or a short footnote. These areas may be interlinked and cross referenced in any manner so as to form a directed graph of text segments through the use of 'branches' (unconditional jumps between two fragments) and 'links' (conditional jumps, a generalization of the manuscript footnote principle)." The user may move through his file by pointing with his light pen at appropriate cross references in a "menu" and returning, if he wishes, to his original place by using the console RETURN key.

As one example of a useful application of the hypertext idea, the researchers cite an encyclopedia or dictionary - something which needs constant updating and which contains many cross references.

The HES internal data structure is not hierarchical, so the facility of exploring text at different levels, as in the AHI system, is not available.

1.b.3 File Retrieval and Editing System (FRESS) [4,5]

This is a commercial development of HES, incorporating also some of the features of the AHI system.

1.b.4 Editing of "flexible" text

Certain of the facilities provided by AHI, HES and FRESS can be described as editing, since they serve to modify the text. In HES, for example, the console has a set of function keys to inform the system which type of edit is to be performed and a light pen to indicate the piece of text involved. Thus deletion is performed simply by pressing the DELETE key and pointing with the light pen to the beginning and to the end of the text to be deleted. The AHI researchers describe [8] five editing operations - Delete, Insert, Replace, Move, Copy - which are associated with "entities", either within the text of statements - Character, Text (arbitrary strings), Word, Visible (print string), Invisible (gap string) - or for structure manipulation - Statement, Branch (statement plus all substructures), Group (sublist of branches), Plex (complete list of branches).

Nevertheless, when text is being thought of as "a dynamic and plastic structure that we continually develop and alter", there is no clear distinction between creating and editing, between editing and using, or even between creating and using. If we are continually developing our text "to represent the current state of our evolving goals ... knowledge ... and data", at what point have we finished creating and begun to edit? When can we be said simply to use it, if we are making continual alterations?

This, then, is the interesting point about editing with such flexible text - that it cannot be thought of as a completely separate process but, rather, as interwoven with all other procedures for manipulating the text. This is well-illustrated in the AHI idea of a user "service system", comprising all the operations, "service functions", a user may perform on his text -

moving to a named statement, examining all statements in one branch, deleting a character, replacing a word, moving a statement, inserting a statement, and so on.

1.b.5 In theory, there is no need for any "final" printed output from such text manipulation systems, but because financial, technical and social obstacles exist to any widespread adoption of purely on-line methods of working, hard-copy output is still important. This brings us to our second area of concern.

1.c Printing and publishing

The use of computers in the production of certain types of books and pamphlets is by now well-established.

The simplest method is to make offset printing-plates from photographs of computer line-printer output, but the limitations of the character set (in both range and quality) of most line-printers may make this completely unsatisfactory, or at least undesirable.

A high standard of typography is possible when the prepared data from computer magnetic tapes is read by the high-speed type-setting machines. The computer memory associated with the type-setter holds instructions for creating the characters of the type fount or founts. These instructions may be modified to vary the letterform - roman, italic, bold or whatever - as required, or to change the type size by the necessary number of points.

1.c.1 Material suitable for computer typesetting

Material which can be economically typeset by computer has input data that needs considerable processing or frequent revision

or that is capable of producing useful secondary information when processed.

A few examples will be given here; others may be found in Bozman,[9] Martin[10] and Phillips[11].

Numerical tables are suitable for handling in this way. The U.S. National Bureau of Standards Experimental transition probabilities for spectral lines of seventy elements is an early and successful experiment, published in 1962. The tables comprised six columns of figures. The data of the first three columns was read from punched cards and the results for the other three columns calculated by the computer.

Most of the major indexing and abstracting publications are also now computer-based, one of the best-known examples being the Index medicus of the U.S. National Library of Medicine. The data, sorted and listed by subject with an author index, is used also under the MEDLARS scheme (handled in the United Kingdom by the British Library Lending Services and the University of Newcastle upon Tyne) where special bibliographies can be prepared for enquirers from the data bank and on-line searching of this bank is possible.

The British national bibliography is computer-typeset after its original data input has been sorted and indexes prepared. The tapes used to produce the bibliography are available for libraries and other interested institutions. The need for cumulative issues from weekly to monthly to quarterly and by stages to annual and then larger cumulations makes this type of material particularly suitable for computer processing.

Oxford University Press are producing concordances to Shakespeare by key-punching the plays and searching the text by computer. The plays themselves may then be computer-typeset, although it would not have been economically justifiable to do this if the concordances were not being produced from the same data. Any catalogues, directories, listings, which need to be updated and reprinted frequently may be considered for computer handling. The publishing firm, Whitaker, now keeps the data for its annual British books in print in machine-readable form.

1.c.2 Editing of data for computer typesetting

It hardly needs to be stressed that the aim in book production, as distinct from the on-line, flexible use of text discussed in Section 1.b, is to produce satisfactory printed output, so that matters of format - type-face, page layout, indention, line justification, hyphenation, capitalisation - become of first importance.

It could be argued that at least some of these should be considered as part of the editing process, since text modification is involved in such things as line justification and hyphenation. Van Dam and Rice state that "text-editing programs ... provide more elaborate formatting functions, allowing for fairly intricate page layout to be performed on-line. In addition, a nontrivial routine is usually provided so that the text may be printed with changeable margins, paragraphs, upper and lower case characters, etc." [4]

In fact, there seems to be general agreement, even in the systems described by van Dam and Rice themselves, to limit the concept of editing to the correcting and updating of text before the final line and page arranging takes place. This may be

because a formatting program is always essential for such text, while a correcting/updating program may not be, and so it is convenient to separate the two concepts. This point is reinforced in the account by Teperman and Katzenelson [12] of their FORMAT EDITOR (not actually concerned with typesetting) with its two modes: a production and an edit mode. The second is used only when changes are to be made and is not concerned with formatting - although it is possible to amend format codes in the text.

In the following outline by Martin, editing would take place, if required, as the last step in the first process:

> "Conceptually, the processes of computer typesetting can be divided as follows:
>
> a Input and creating of files of text and format control instructions
>
> b Line and page formatting
>
> c Output to a composing machine drive tape, with the required control codes." [10]

It is perhaps not clear from Martin's description that the format control instructions would almost certainly be added at appropriate points in the original text to form part of the stored text file and so could be amended by the editor.

The editor, then, is one of a suite of programs which may be used to produce the final output tape that will control the printing. A number of special projects, such as MEDLARS (mentioned in Section 1.c.1) and the Computer Typesetting Research Project at the University of Newcastle upon Tyne (see below) have developed their own suites of programs.

Commercial printers are unlikely to be willing or able to undertake this and so make use of programs written by computer manufacturers "to achieve a limited result with a wide application that will sell their equipment". [11] The editors for these systems are workable but unremarkable. One of the best-known is the IBM TEXT 360 [11] which can insert and delete lines and copy lines from the original text in some new order, but which has few facilities for manipulating individual words or phrases.

The editors for special projects tend to be rather more sophisticated, since each is catering for the known needs of one particular application rather than the unknown needs of possible applications. An interesting account of the batch editor in the Newcastle Computer Typesetting Research Project has been given by M. D. Poole, [13] referring particularly to the facility for changing each occurrence of a piece of text throughout a specified area of the file.

1.d Internal and external representation of the text file

Almost nothing has been said so far about the internal storage data structure of text, although this is very relevant to editing.

Where the text is externally simple in form, as, say, that of a FORTRAN program or of some lists and indexes, where changes will be insertions or deletions within lines or insertions or deletions of complete lines, then it may be conveniently stored and dealt with line by line. In other words, a line of a specified length is the basic data unit for input, manipulation in memory, display (if an on-line

system is used) and output.

With the "free-form" text of most books, journals, newspapers, however, the basic unit is the paragraph, which may be of any length, as may insertions and deletions within it. The input line, the display line and the output line are units suitable for the physical devices that handle them and have no logical relationship with one another. To allow editing operations, the storage unit must be arbitrarily large - or, at least, appear so to the user. In practice, it is unlikely that the whole unit will be in main memory at one time; it will almost certainly be divided into "pages" possibly with enough free space left on each page to allow for most insertions, as Deutsch and Lampson describe in their account of the implementation of the QED editor.[14] They report that it is not often necessary to redo the paging because of very large insertions or deletions.

As already mentioned (Section 1.b.1), the AHI text has an internal and external tree structure. No restriction is placed on the length of its basic unit, the statement, stored on pages. If there is no room for an insertion within a statement, it can be copied to another available page and its pointers altered on the tree structure. HES stores text in variable-sized pages, each one being paired with an "order code" page containing text pointers and formatting and structure codes. Editing involves changing pointers and codes, not the text itself. There are limits on the length of the character strings concerned. These are removed under FRESS, where changes are made directly, by rearranging the text within pages.

Van Dam and Rice have suggested the term, "program editor", for an editing program handling line-oriented text (because computer programs are the most common data) and "text editor" for one dealing with free-form text. It will be convenient to adopt these terms here, for want of better. These authors rightly point out, however, that "it is entirely possible (and in fact, quite common) to use a given editor for either text or program editing".[5] For instance, the use of an end-of-line marker can preserve the original line structure of program text even if a text editor is being used. Again, although a program editor making a few changes in "normal" text may transform a newspaper sentence such as

> It is, undoubtedly, quite
> extraordinary that De Gaulle
> should have paid a full-scale
> state goodwill visit to such a
> country.

into

> It is, undoubtedly,
> odd that De Gaulle
> should have paid a full-scale
> state goodwill visit to that remote, unfriendly
> country.

it would still be possible to achieve an acceptable result by later formatting - assuming that the insertion in the fourth line had not, in fact, caused the line to overflow, in which case the rightmost characters might have been lost.

This illustrates what is perhaps the main difference between the two types of editing from the user's point of view. When using a text editor, he has already given instructions about the format of the output and so can make insertions and deletions without having to concern himself again about the effect on the text layout. With a program editor, on the other hand, the

responsibility is the user's to see that altered or inserted lines are in acceptable form.

Another important limitation of the program editor is that a search being made to identify a specified character string will normally be confined to the line, so an occurrence of the string that is divided between two lines will not be found.

1.e Restriction on provisional definition of editing

Since computer programs form the text which programmers most frequently want to amend, it is perhaps not surprising that some of the most interesting and sophisticated editors have been written with program text in mind. Because of this - and because the editing program to be described is itself line-oriented - detailed discussion of editors from now on in this paper will be confined to program editors. (It should be added that some program editors can be used only for computer programs - in specified languages - and include syntax-checking among the facilities provided. These will not be considered here.)

Our definition of editing originally adopted (Section 1.a) is basically unchanged, but its extension is limited. In order to show the further restrictions on our use of the term, a consideration of two other ways of modifying text will be helpful.

CHAPTER 2

EDITORS, MACROPROCESSORS AND STRING-MANIPULATION LANGUAGES

The ways in which text is handled by the three processes are broadly compared in the following pages.

2.a Deficiency of provisional definition of editing

One defect of our definition of editing (Section 1.a) is that it could serve to describe other things. It could, for example, be accepted as a very general description of macroprocessing. Should we then assume that macroprocessing is a subset of editing? The account of the ML/I macroprocessor [15] as being "fed some source text. It performs some transformations on this text and generates some output text ..." [16] could equally be applied to editors - or indeed to some operations of string-processing languages. S. R. Bourne has suggested that "The differences between an editing program, a macrogenerator and a string manipulation language like SNOBOL are in principle quite small. They all enable the user to specify textual alterations to be made to a document and in some cases their functions overlap (e.g. replace a given string of text by another throughout a document)". [17]

2.b Macroprocessors

The differences and resemblances between editors and macroprocessors can best be established by considering the general characteristics of the latter.

2.b.1 General features

The type of alteration for which they are primarily designed is the expansion of the text in a systematic way, previously defined, triggered by a comparatively simple piece of text, called a macro call. This may be a straightforward replacement. For instance, in TRAC [18] terminology we might define a macro by

```
#(ds,TEXT,THIS IS A PIECE OF TEXT
          REPLACING A LATER CALL
          OF THE MACRO TEXT)
```

From then on, the simple expression

```
#(cl,TEXT)
```

when encountered in the input data, will be replaced by the three given lines. Greater variety and elaboration may be provided by replacing LATER by a marker

```
#(ss,TEXT,LATER)
```

which may then be replaced as required at each call. The two calls

```
#(cl,TEXT,FIRST)

#(cl,TEXT,FINAL)
```

would generate respectively

```
THIS IS A PIECE OF TEXT
REPLACING A FIRST CALL
OF THE MACRO TEXT
```

and

```
THIS IS A PIECE OF TEXT
REPLACING A FINAL CALL
OF THE MACRO TEXT
```

Higman [19] has illustrated the use of macros by referring to several children's songs where successive verses remain largely

unchanged, except for one or two words each time. We might give a less neat, but brief, example by defining in GPM [20] terminology a macro for easy production of some of the nonsense verse of Edward Lear:

```
§DEF,LEAR,<
There was an Old Man ~1, who ~2;
When they said, ~3,he replied, ~4,
That ~5 Old Man ~1.
>;
```

The formal parameters in this macro are replaced by real ones as wanted, as in

```
§LEAR,in a casement,held up his hands in amazement,
Sir - you'll fall,Not at all,incipient;
```

or

```
§LEAR,of the West,wore a pale plum-coloured vest,
Does it fit?,Not a bit,uneasy;
```

A slightly more plausible use of a macrogenerator may be suggested by the following form-letter:

```
§DEF,FORMLETTER,<
Dear ~1,

     We are sure that if you scan our magnificent
gift catalogue, you will find exactly what you seek
for your ~2's ~3 present.
     There is no doubt that ~4 will be delighted
with the quality of the article you choose for
~5.
     We look forward, ~1, to receiving your order.

                         Yours faithfully,
>;
```

The firm could then produce the required letter by

```
§FORMLETTER,Madam,husband,Christmas,he,him;
```

or greater familiarity with the customer might inspire

```
§FORMLETTER,Mrs. Smith,daughter,twenty-first birthday,she,her;
```

Although there could be applications such as the above in business and advertising, macroprocessors are clearly best suited

for the purpose for which most of them have been designed - that is, for "extending an existing programming language, called the base language, by introducing a new syntactic unit which is described in terms of the existing syntactic units of the base language". [16] Even although the expression "general purpose" has several times been used to describe such a macroprocessor (ML/I, [15,16] Strachey[20]), Waite's term, "language-independent"[21], is more explanatory.

The need for a new syntactic unit is most likely to be felt when dealing with a low-level language, but it is not impossible that a FORTRAN programmer may want to use, say, a multiple assignment statement. He might then define an ML/I macro, MA, to generate a variable number of assignment statements setting each argument (separated from others by a comma) on the left hand side of an equals sign equal to the argument on the right hand side:

```
MCDEF MA Nl OPT , Nl OR = ALL NL
AS<  MCSET T3 = 1
%L1.    %AT3.=%AT1.
MCSET T3 = T3+1
MCGO L2 IF T3 = T1
MCGO L1
%L2.>
```

This would enable him to write in his program some such statement as

```
MA I,J,KK=NNN
```

which would be expanded by the ML/I processor into

```
I=NNN
J=NNN
KK=NNN
```

The use of a template macroprocessor like LIMP [21] would allow a more "natural-looking" call:

```
I,J,K,L=N
```

Such elementary examples as those above do not do justice to the power and capabilities of macrogenerators like GPM, TRAC, ML/I and LIMP, which are programming languages where the definition of a macro may employ such facilities as branches and loops, arithmetic operations, recursion and nesting, but they serve to show the basic principles underlying the operations of macroprocessors and enable us to see the differences between them and editors.

2.b.2 Essential differences between macroprocessors and editors

The editing commands of an editor are equivalent to the macro calls of a macroprocessor. The editor identifies each command and carries out the appropriate previously-defined operation, substituting where necessary actual parameters for formal ones, in the same way as the macroprocessor handles its calls.

The essential difference between the two lies in the position in the system of the text file which is being modified. For the macroprocessor, the text and the macro calls form a single logical input file. It accepts a line of text, searches it for macro calls - both system and user-defined - and outputs it unchanged if no macro call is found. If a macro call is identified. it is replaced in the text file by the result of the call - usually, as we have seen, a larger piece of text.

For the editor, the command file and the text file are quite separate entities, the latter being moved and altered in accordance with the operations specified in the former. Even when editing commands are interspersed with text in the same physical file - as with some primitive editors - they are logically distinct. Lines of text can be read only if editing commands are given to cause this action and these lines will not be examined in any

way unless this has been specified.

As a simple example, we may assume an editing command "Copy 1" which directs that one line of text be read from the input file and written to the output. This would be obeyed without any examination of the line. With the equivalent TRAC command "#(ps,#(rs))", the line would be scanned by the processor to see if it contained any further TRAC functions.

2.b.2.a Assuming that the structure of the file to be edited can be used as data for the editor, an edit can always be performed on it. The user may well not have known of the editor when he created his file - or he may have hoped that he would not need to use it!

2.b.2.b On the other hand, it is generally true that the data file for a macroprocessor must be specially created to serve as input to it, since macro calls are necessarily part of it, and that it is not possible to effect changes in a text file originally produced for some other purpose.

Some exceptions may be cited. Since the ML/I processor allows macro calls without the initial special symbols ("§", "#(") of GPM or TRAC, it is possible to carry out some replacements in "normal" text by defining such macros as

```
MCDEF KING AS <QUEEN>
MCDEF HIS WITHS MAJESTY AS <HER MAJESTY>
```

If the file is then handled by the macroprocessor, all occurrences of "KING" will be replaced by "QUEEN" and all occurrences of "HIS MAJESTY" by "HER MAJESTY".

The LIMP processor, also, would permit such substitutions, as it recognises macro calls by testing each line against a template. A line containing "KING" could be recognised as

matching the template

* KING *

with any characters (or the null string) before "KING" being interpreted as the first actual parameter and any after as the second. The necessary replacement could be achieved if the macro definition had been, say,

END PR = 10 'QUEEN' 20 /

with the zero indicating that the parameter has to be printed unchanged.

It would even, in fact, be possible to carry out the same change with TRAC, if advantage were taken of its power of recursion to define a function that would repeatedly call itself - reading and altering, where necessary, a line at each call, until the end of the file.

For example:

```
#(ds,A,(#(ds,LINE,#(rs))#(ss,LINE,KING)#(ps,#(cl,LINE,QUEEN))
                                          #(cl,A)))#(cl,A)
```

The text file could then be input after the TRAC function, even although the file itself contains no TRAC functions.

In these instances, however, only a very limited use is being made of the abilities of the macroprocessors concerned. When they are being profitably used, the assertions at the beginning of this section (2.b.2.b) hold true.

2.b.3 Although the application of ML/I to text editing is mentioned once or twice by Brown,[15,16] it is clear from his use of the expression "systematic editing" and his example of converting from FORTRAN IV to a dialect of FORTRAN II, that he is not thinking of text editing in the sense of making random

changes and corrections. Waite [21] suggests the possibility of incorporating most of the features of LIMP into a general text editing system "thus combining the tasks of text correction and expansion", but details of the research have not yet been made public.

2.c String-manipulation languages

The relationship of a string-manipulation language to its data is similar to that of an editor rather than to that of a macroprocessor. Given a text file on which operations are to be performed, it is possible to write a program in any language (capable of performing the operations) to do this - the data file need not have been created with a particular language in mind. It could even be data prepared for a macroprocessor and the string language program could simulate the action of the macroprocessor. For instance, the multiple assignment macro of Section 2.b.1 might be implemented in SNOBOL3 [22,23] by

```
1   SYSPIT  *LINE*     /F(END)
    LINE  ANCHOR()    '      MA ' *VARS* '=' *VAL* ' '  /S(2)
    SYSPOT  *LINE*     /(1)
2   VARS  *VAR* ',' *REST* = REST  /F(3)
    TEMP = REST
    SYSPOT  '       ' VAR '=' VAL  /(2)
3   SYSPOT  '       ' TEMP '=' VAL  /(1)
END
```

Unlike macroprocessors and editors, string-manipulation language programs do not necessarily modify text - they may count the occurrences of words or prepare indexes, for example. (Of course, macroprocessors and editors may have additional features which perform operations other than text modification - compare Benjamin's example, [24] illustrating the use of a macro

facility within an editor, which establishes the number of occurrences of a string - but these are not an essential part of their structure.)

2.c.3 Like macroprocessors, string languages are used most efficiently when the same operations - with perhaps slight modifications - are repeatedly performed on different data. They, too, can readily generate the "form-letter" type of output, discussed in Section 2.b.1, by use of procedures with variable parameters. (Good examples are given by Barnett.[25]) The same results could be achieved by editing programs only be repeated edits of the file, specifying the alterations each time. Bourne's example (Section 2.a) of the operation of replacing one string by another throughout a file, is elementary for macroprocessors and string languages, but fairly sophisticated for editors.

2.d Comparable features of the three processes

2.d.1 The pattern-matching ability implicit in such a replacement operation is shared by most editors with string languages and macroprocessors. A possible editing instruction

```
REPLACE/BOY/GIRL
```

is similar to the COMIT [26] rule

```
* ∅ + BOY + ∅ = 1 + GIRL + 3 //* ...   *
```

and to the combination of the TRAC functions

```
#(ss,string,BOY)#(cl,string,GIRL)
```

All test for the occurrence of "BOY" and, if it is found, replace it by "GIRL".

Few editors go any further than this in pattern-matching - but not all string-manipulation languages go so far. In SNAP [25],

for instance, the search for "BOY" would be made in lower-level terms. On the other hand, the MITEM6 editor [3] uses even more advanced pattern-matching than COMIT or SNOBOL3 and incorporates several of the features of SNOBOL4.[27,28] (MITEM6 is discussed in more detail in Section 4.e.)

2.d.2 The ability to name strings, while not provided in COMIT, has proved generally helpful and is found in later languages such as SNOBOL and SNAP and in several macrolanguages like GPM, TRAC and ML/I. The "distributed" names of such template-matching macroprocessors as LIMP do not serve the same purpose of easy reference. Several of the more sophisticated editors provide this facility in a limited way - the buffers of QED [14] and of the higher versions of MITEM, the eight STRINGS of MTS *Ed. [29]

2.d.3 Concatenation and deconcatenation of strings, standard operations for string-manipulation languages, are not performed by editors.

2.d.4 In general, editors are not good at transposing sections of files - but this is often a clumsy operation even in a programming language. To move a line or two of a file to a later position in the file can be accomplished very easily by a macroprocessor or a string-processing language (or by the QED and MITEM editors mentioned above) merely by storing the line or lines under a particular name and referring to that later when needed, but to take sixty lines from near the end of a two thousand line file and insert them nearer the beginning means a considerable amount of shuttling backwards and forwards, rewinding or back-spacing. The DOLPHIN editor [30] has an elegant solution to the problem, which is referred to in Section 4.d.1.b. Those editors which allow switches of input and output streams during editing

(the University of Cambridge EDIT [17] and MITEM, for example) enable the user to do a certain amount of transposing of blocks of lines.

2.e Distinguishing characteristics of editors

Editors, then, share some of the features of string-manipulation languages and of macroprocessors. They differ from them in being designed primarily to make small, single, unsystematic changes in text - operations which cannot be efficiently performed in a programming language, since a program written to effect such arbitrary alterations in one file would, presumably, never be of use again. (It is interesting to note that Forte, in an example in his SNOBOL3 primer,[23] suggests a primitive editor, written in SNOBOL, to make such changes, not a SNOBOL program to make them directly. It is also interesting that the primitive editor is, in fact, a macroprocessor.)

Editors are not programming languages - although some of them have limited programming abilities - and so are, in general, unsuited to systematic and repetitive text manipulation. As was stated by Deutsch and Lampson [14] in a report on an early editing program, "small programs can readily be written in SNOBOL, or other string-processing languages to accomplish repetitive editing operations". This is emphasised in a more recent article by Benjamin [24] : "For the most complicated text manipulation problems, specialized languages such as COMIT, LISP or SNOBOL would be utilized."

2.f Revised definition of editing

We may now submit a revised definition of editing - for our purposes in this paper - as that process whereby arbitrary changes, or simple repetitive ones, may easily be made to machine-readable text, for which the basic internal and external data units are the same.

CHAPTER 3

THE DESIGN OF AN EDITING PROGRAM

The main factors that determine the basic structure of an editor are examined in this chapter.

3.a On-line and background editing

If a sophisticated alphanumeric CRT display unit is available, the editor will enable the user to press a key indicating the required operation (deletion, insertion, replacement, copying, for example) and to indicate on the screen, with some such device as a light pen, cursor or "mouse", the point or points affected in the text. Since such systems are still uncommon - and not of direct concern to our present purposes - they will not be considered further.

In all other cases, whether the editor is used interactively or in batch processing, an editing command language is necessary. Since there is no "delete" key to be pressed, the operation has to be specified in some other way - it may be by the use of the word "delete" or by an abbreviation, perhaps "del", or by a symbol, possibly "-". If the piece of text to be operated on cannot be pointed at, another way must be found of indicating it. For instance, 5.13.27-12.06.08 could mean the area of text from the 27th character on line 13

of page 5 to the 8th character on line 6 of page 12 and COW/MOON the area from the first occurrence of the word "COW" to the first following occurrence of the word "MOON".

If any kind of CRT unit (even relatively unsophisticated) is to be used, the editor will give the user the power to display at one time as many lines of text as the screen will accommodate, in order that he may check the position he has reached in his file and may consider his next editing operation.

If a teletype terminal is to be used, probably only one or two lines of text could be conveniently printed there for display at any one time, because of the much slower speed. In this case, it would be almost essential for the user to have beside him a listing of the text he is editing.

An editor intended for interactive use will interpret and obey each command immediately and supply explanatory messages to the user at his terminal. The failure of a command need not bring an editing session to an end, since the user may be able to correct the error. It is possible, too, that the program may allow him to "wander" backwards and forwards through his file.

With a background editor, however, it is more efficient that all commands be read, possibly sorted and checked and then executed in a single pass through the file. Explanatory messages are less important, although reports of failure are still necessary. The action to be taken in the case of failure must be rigorously defined, since the results of an accumulation of errors could be very serious. With most failures, it would, in fact, be necessary to abandon the editing job to avoid the possibility of such an accumulation.

3.b Single file or creation of second file

The editing facilities provided will depend on whether a completely new file is being created as editing proceeds or whether the operations are being performed on a single file.

In the former case, deletion is a simple matter of reading the old file without writing to the new and insertion simply writing to the new without moving the old. It is almost certain, however, that the editor will require that amending should be done sequentially, otherwise a return to an earlier point in the file would involve first copying the remainder of the old file to the new, then going back to the beginning and copying the new file (now considered the old) up to the point to which the user wants to return - a rather inefficient operation, although quite feasible. With sequential processing, this two-file implementation allows the editor to supply commands for the on-line user to recover from an error, as it is simple to cancel the effect of a command simply by returning both files to their position before its execution. It may also be convenient in some commands to define areas of text by line numbers, or by a combination of page and line numbers.

If only one file is used, editing need not be done sequentially and the user may be enabled to move backwards and forwards at will. Unless a CRT display is used, however, it may be difficult for him to keep track of all the changes he has made and to find his way back (even although he presumably has beside him an annotated listing of his original file), so it is quite common even for one-file editors to require sequential processing - for instance, the DOLPHIN [30] and COTAN [31] editors.

If so, they may well provide commands using line numbers. This is less likely, but still possible, when movement either way is allowed. Even although deletion and insertion may have altered the position of lines in the file relative to the start, the original lines may well have been allotted numbers by the program which do not change throughout the editing session or the filing system to which the editor belongs may have its own numbering - as does the MTS.[29] It is also possible that in the implementation of the editor, the text is not actually being rearranged in memory immediately, but that links to a "scratch page list" (the Brandeis University term)[24] or "patch block" (the DOLPHIN expression) of amended portions of text are being inserted, to produce the revised file in sequential form only at the end of the editing session.

The cancellation of an unwanted result is not so easily achieved in a single-file editor, but a sophisticated one may provide for it by keeping a record of the last few operations performed along with a copy of the original pieces of text altered or deleted. The QED editor[14] does this to a limited extent. Where the user can move backwards through his file, however, there is less need for such a facility.

3.c Addressing points in a file

3.c.1 Addressing by line number

If a line-numbered listing of the file is available - and with some systems, for instance the Newcastle Computer Typesetting Research Project,[13] this is always so - the editing program will run more efficiently and the user's task be made simpler and less liable to error, if use is made of these numbers for addressing.

If a user wants to delete a section of his file from, say, line 169 to line 203 inclusive, then by far the easiest and least ambiguous way to refer to the section is by these numbers.

There are, however, arguments against the use of line numbers. If some editing is done sequentially on the file and another pass made through it for a second editing session, the line numbering may well have changed. If editing is not being done sequentially, then, as already mentioned (Section 3.b), line numbers are unlikely to prove satisfactory - unless they are fixed ones.

Only an editor designed for restricted use within a system where line-numbering is accepted (such as the above Newcastle one) could depend solely on line number addressing.

3.c.2 Addressing by context

Reference by context - that is to say, finding a particular point in a file by searching for a given string of characters - can always be used, whether or not there are difficulties about line numbering. For that reason, nearly all editors employ it, in one or both of its forms: the SNOBOL concept of the "anchored scan", where the search is limited to the start of each line, or the more general search throughout each line. Although the former is considerably more efficient than the latter, it is still more demanding on CPU time than counting lines.

Another disadvantage of the procedure is that it is easy for the user to make a mistake in specifying his context. He may think that the first or only occurrence of string "XYZ" is halfway through his file, but there may be an earlier one that he has missed.

3.c.3 Addressing by either line number or context

An editing program that allows addressing by either line number or context may be thought to give the user the best of both worlds, but the price to be paid for this is usually a more difficult editing language - discouraging for the user - since the number of commands is necessarily increased and the choice of neat, mnemonic notation consequently less easy.

Some editors permit addressing by combinations of line numbers, context and/or integer displacements. Under the EDITDC [32] sub-system of KOS (Kent On-Line System), the address

```
23 /ABC/ + 2
```

would find the second line beyond the first line after line 23 to begin with the string "ABC".

Under QED [14]

```
XYZ -3
```

would find the third line before the next line containing "XYZ".

3.d Pointers to the file

All editors use (although some stress it more than others) the concept of a line pointer which indicates the line currently being addressed and which moves (usually forwards, but also backwards with some editors, as we have seen) when another line reference is given by a line number or context command.

In addition to the line pointer, some programs have a character pointer which may be thought of as moving along the current line, in accordance with editing commands. Backward movement is often allowed, as this is easy to implement when the line is in memory.

A good example of an editor with a character pointer is the COTAN Amender.[31] Nearly all its commands are duplicated, with only a single character different to indicate whether the directive is a line one or a character one. Thus "CH.XYZ" would leave the line pointer at the beginning of the next line starting with "XYZ", and "CB/XYZ" would leave the character pointer at the first character of the next occurrence of "XYZ" in the current line, while "C6" would move the line pointer on six lines and "C6/" the character pointer six characters.

In a few editors, notably the widely-used ICL EDIT facility for the 4100 series,[6] the line and character pointers become one. In this case the editor is dealing with the data character by character and not line by line, although the idea of the current line is still used. The EDIT command to find a line beginning with a character string - "FLstring" - leaves the pointer at the character following that string.

3.d.1 While the line pointer concept is essential to the working of the editor (since the user would not be able to continue editing if the file position were undefined after each editing operation - unless repositioning by line number were part of each instruction), there is little to choose between using a character pointer and doing without one.

An example will illustrate the different types of command needed in the two cases.

The COTAN Amender, using a character pointer, could correct the following line

SEESON OF MITS AND MIELLOW FRUFUITFULNESS

by the sequence of commands

```
CA/E     Copy to after the first E
R/A      Replace the next character by A
C9/      Copy a further 9 characters
I/S      Insert S
CB/I     Copy to before the next I
D1/      Delete 1 character
CA/U     Copy to after the next U
D2/      Delete 2 characters
```

while the Cambridge editor,[17] not using a pointer, would use a sequence such as

```
E/EE/EA/  Erase the first EE found and replace with EA
E/FU//    Erase the first FU found and replace with null
A/I/S/    After the first I insert S
E/IE/E    Erase the first IE and replace with E
```

In the second case the commands are longer, since a positioning string is needed with each one, but they are fewer in number. It does not matter in what order they are given. With the former type of editor, the order is of the utmost importance. Even when such editors allow backward movement within the line (which COTAN does not), the user has to be constantly aware of his precise position.

The pointer is not, in fact, absent in such editors as the Cambridge one. It is reset to the beginning of the line before each context search is made (thus using more CPU time than the other method) and the user has to concern himself with it to the extent that the context string he selects to identify the location of his change must be the first occurrence of the string. In the example above, it would not have been possible to delete the "I" in MIELLOW" by

```
E/MI/M/
```

since there is an earlier occurrence of "MI" in "MISTS". Such a pointer, however, is unobtrusive for the user.

The QED and Cambridge systems provide alternative ways of operating on individual lines, with a pointer or without. When a pointer is being used (for on-line editing at a teletype), the directives are considerably more clumsy than those of COTAN. Deutsch and Lampson write regretfully, "There are, unfortunately, no mechanisms for addressing characters within a line on a teletype which are not more trouble than they are worth".[14]

3.d.2 There may well be occasions when the character pointer idea is useful, particularly when amending data in tabular form and when the editor has a facility for repeating corrections. For instance, the sequence of COTAN directives

```
LOOP
C8/
D4/
N
ENDLOOP
```

would delete columns 9-12 of each line of the file. Editors which do not use a character pointer would be unable to achieve the same result without specifying the actual characters to be deleted in each line. In most circumstances, however, it seems a pity that the user should have to take account of a character pointer position. As M. D. Poole puts it, "Within a line, counting of words or characters is tedious and unreliable ...".[13]

3.e Range of operations provided

Apart from the general considerations discussed in the four previous sections, the operations provided in an editor will depend on whether it is designed for use at many installations or to satisfy the known needs of one environment, where the type of user, the other facilities in the operating system, the kind of files to be amended, can all be taken into consideration.

In the second case, some very simple editing system may suffice. For instance, the University of Bradford BESS editor [33] and the University College of Wales Disc-File Editor [34] provide only the three basic operations of inserting, deleting and replacing complete lines, with addressing by line number. Although the KOS EDITDC [32] has more flexible means of addressing, referred to in Section 3.c.3, and various instructions controlling output, it, too, performs only the three operations on complete lines. The edit facilities of the University of Strathclyde UNICORN Filing System [35] are limited to the two essentials of inserting and deleting of lines, with addressing only by context (at the start of a line) and by advancing a specified number of lines.

Most editors will include, also, means of deleting, inserting and/or replacing characters within lines, to allow small changes to be made easily.

Since the user may well want to repeat a correction - or several corrections - throughout a line or a section of the file or the whole file, some editors make provision for this. Just how far to go in this matter is one of the more interesting decisions to be made in designing an editing system. To increase the complexity of an editor to enable it to perform tasks for which a programming language is essentially suited seems pointless, yet a user who has other editing to do on his file is not likely to want to write a program for macroprocessor or string-manipulation language as well - even assuming that one is available at his installation.

Other operations which may be included in the editor are methods of recovery from errors (in on-line systems), switching

of input and output streams, varying of line length, storing of commands or of sections of text for later use, tabulating or other simple formatting facilities. These points, as well as those in earlier sections, will be illustrated and developed in Chapters 4, 5 and 6.

3.f Choice of an editing language

The aim of most editing systems is to find a brief (to reduce typing) mnemonic name for each type of operation. That is then used, with one or more parameters - integers and/or character strings - where appropriate, for an editing command. The initial "D" for example, is frequently used to indicate the delete operation, so that the BESS [33] instruction

```
O81DO5
```

specifies the deletion of five lines, starting with line no.81, while in the MTS File Editing Program, *Ed, [29]

```
D /XYZ/
```

will delete the next occurrence of "XYZ" and

```
D 12
```

will delete twelve lines starting with the current one.

Where only a few operations are provided, the editing language may be very simple indeed and may well consist of the three directives "D[elete]", "I[nsert]", "R[eplace]". The greater the choice of operation, the more difficult it is to find a notation easy for the user to follow and to remember.

Apart from the delete operations mentioned above, other possible ones are deleting from one string to another, from the current line to line no. n, from the current line to the

line beginning with (or containing) a certain string, from the current line to the end of the file. Since the letter "D", will not serve to define all such variations, other letters or symbols have to be introduced and the mnemonic value of the language becomes harder to maintain.

This is particularly so when an attempt is made to keep mainly to one-character directives. Of the two "delete" commands in the University of Cambridge EDIT program, one uses D and the other the minus sign; of the three "insert" commands, one uses I and the others A and B respectively; of the three "replace" commands, one uses R and the others E and G.

The alternative method of adding characters to the basic mnemonic is comparatively easy to follow, even although it requires more typing. This may be shown by the COTAN directives:

D6	Delete next six lines
DA.XYZ	Delete from current line to after line beginning XYZ
DB.XYZ	Delete from current line to before line beginning XYZ
DAL	Delete from current line to after last line
DBL	Delete from current line to before last line
DELALL.XYZ	Delete all lines beginning XYZ

This last command illustrates a fairly common characteristic of editing languages. Whereas frequently-used instructions are left as short as possible, concise forms are not necessarily sought for the others. Thus we have ENDEDIT (KOS EDITDC), RESET and FINISH (MINIMOP editor), [36] STOP, START, LOOP, ENDLOOP (COTAN Amender).

In some instances, commands are implied rather than stated explicitly in letters or symbols. A valid instruction in several editors would be

Unfortunately, there is no consistency in interpretation. To the COTAN Amender it would mean "Copy the next 50 lines"; in KOS EDITDC, "Copy up to line no. 50" and in the SETUP editor of the CDC 6000 series, [37] "Delete line no. 50".

3.f.1 Quite different from the form of notation generally used by editing programs up till now, is that recently suggested by Moudrý. [38] This involves specifying each editing directive in the form of a statement with a left hand side ("the specification of the error") and a right hand side ("the correct text") separated by an "equals" sign. The following examples will give some idea of the simplicity of the notation:

```
'I WANT TO CHANGE THIS'=/I LIKE THIS BETTER/
```

The text delimited by the solidus replaces that delimited by the apostrophe.

```
131=?THIS IS A NEW LINE.?
```

The text delimited by the question mark replaces line no. 131.

```
28/CAN'T/=/CANNOT/
```

The first occurrence of CAN'T in line 28 is replaced by CANNOT.

```
/ABC/../LMN/=/REPLACEMENT TEXT/
```

The text from the next occurrence of ABC to the next following occurrence of LMN is replaced by REPLACEMENT TEXT.

```
/ABC/)..(/LMN/=/REPLACEMENT TEXT/
```

The text between the next occurrence of ABC and the next following occurrence of LMN is replaced by REPLACEMENT TEXT.

```
201=          Delete line no 201

201..206=     Delete lines no 201-206
```

```
201)(202=/   INSERTION TEXT   /
```

Insert line delimited by the solidus between lines no. 201 and 202.

```
317'I'..'I'..'I')'I'='EI'
```

The fourth occurrence of I in line 317 is changed to EI.

With the use of only four symbols (the equals sign, the double full-stop and the left and right hand brackets) a powerful editing language can be developed. It fully justifies Moudrý's claim that it is "easily readable and easily understood". It is reminiscent of the notation of the DOLPHIN editor, which is to be examined in more detail in Section 4.d.

3.f.2 The question of an editing command notation will recur throughout Chapters 4 and 5 in discussion of particular editors.

CHAPTER 4

FIVE PROGRAM EDITORS

In this chapter, some of the more interesting points of five program editors are discussed in order to illustrate more clearly a number of matters referred to in the last chapter and to introduce certain special features not hitherto mentioned.

4.a EDIT: a program for editing text on the Titan computer at the University of Cambridge.[17,39]

The program, EDIT, may be used off-line, although it is designed primarily for interactive use with a teletype. Either line numbers or context may be used for addressing. Editing proceeds sequentially through the file, a new file being created in the process. It is, however, possible to return to an earlier point in the file being edited by specifying its line number.

The concept of a character pointer may be used but is not essential.

Directives are almost all single characters.

4.a.1 Interesting points

4.a.1.a Although the idea of string buffers is not employed, there is a command (indicated by an apostrophe) that causes the last context command (that is one with a string parameter or parameters) to be obeyed again without its having to be retyped.

4.a.1.b By the use of streams, EDIT can achieve a considerable degree of text manipulation. Files may be merged by connecting each of them, when appropriate, to the input stream. A block of text may be moved to a later point in the file by writing it to a temporary output stream and inserting it when wanted. The command

H69 O3

would insert the lines from output stream 3 before line 69 of the file being edited.

A looping facility can be achieved by writing a sequence of commands to a stream and making that the command stream when needed. As will be mentioned later, this facility is limited.

4.a.1.c A listing of a file, with line numbers added, may be produced with the command

PmLn

which lists input stream, m, on output stream, n.

4.a.1.d EDIT has current line commands to replace one string by another and to insert a string before or after an existing string. To provide for cases where there might be difficulty in identifying a particular string in the line, the character pointer notion is introduced. Associated with this are three commands; one which moves the start of the edit field one character to the right, another which erases a character, and a third which resets the edit field to

the beginning of the line.

4.a.2 Defects

4.a.2.a Because of the single-character directives, their mnemonic quality is necessarily rather strained. For instance, of the three "Replace" commands, one is R, one E and the third G. The three "Insert" commands are I, A and B; the two "Delete" ones, D and -. (This has been referred to in Section 3.f.) This editor too, shares with several others the notion (possibly first used in the IBM 360-67 CP/CMS editor) of "locating" a line containing a certain string and "finding" a line beginning with a certain string. Since the two terms might well have been interchanged, the user has to find his own mnemonic to distinguish L from F.

4.a.2.b There is no direct command to delete a particular string, although this may be done by replacing it with the null string in a replacement command; nor is there any direct way of deleting from the current line up to another line, specified by context. The line number, or the number of lines, has to be given. (The result could, however, be achieved by using the stream facility and writing the lines to be deleted to a temporary output stream.)

4.a.2.c There seems to be no satisfactory way of setting up a looping sequence of commands. The sequence could be written on a stream which could then be connected temporarily as the command stream, but there is apparently no way of calling this n times without giving the call command n times.

A repetitive facility is, however, supplied in the G command which replaces one string by another throughout the file.

4.a.2.d EDIT handles lines of up to 72 characters, the width of the teletype carriage. Lines longer than this are divided into 72-character "part lines". A character string will not be recognised if it is divided between the two parts of a line. With some files, this might be a serious defect.

4.a.2.e Although editing may be cancelled if something has gone seriously wrong, there is no method of correcting the result of a small error without finishing one editing session and beginning another with the new file.

4.b COTAN: a multi-access on-line system, produced by the Computing and Applied Mathematics Group of the Culham Laboratory in Abingdon.[31]

The COTAN editor, called by means of the "AMEND" command, is an example of an easy to use and fairly commonly used program editor with no really sophisticated facilities.

It employs the concept of a character pointer moving across the line as well as the usual line pointer.

Although its use of "Copy" commands implies that a new file is being created during editing, operations are in fact being performed on a single file.

Directives apply to the current line or to later parts of the file. By means of a "START" command, however, it is possible to return to the beginning of the file.

Directives are mainly one, two or three letter mnemonics (such as DA for "Delete to after" and DBL for "Delete to before last") but some are longer (DELALL for "Delete all" and PNEW for "Print new lines", for example) and some are complete words (such

as STOP, START, LOOP). This editor has been referred to several times in Chapters 3, notably in Sections 3.d and 3.f.

4.b.1 Interesting points and defects

To some extent the good, flexible points about the COTAN amender are also its weaknesses.

4.b.1.a Where possible, the line and character directives are identical, except for a terminating character. The commands

```
CB.XYZ
CB/XYZ
```

mean "Copy to before line beginning 'XYZ'" and "Copy the current line to before 'XYZ'", respectively.

While there are obvious advantages in this, there are obvious dangers too. As the User's guide warns, the accidental omission of the solidus in

```
DAL/
```

would cause the rest of the file to be deleted instead of the rest of the current line, because the terminator in line directives may be left out if not needed as a separator between directive and parameter.

4.b.1.b In several cases, the user has the choice of copying (or deleting) to before or after a particular character string (or the line beginning with it). In the same way, there are "before" and "after" commands referring to the last non-blank character in the line and to the last line of the file.

This certainly allows him to define his object in the way most immediately obvious to him, but it is doubtful whether this outweighs the disadvantage of having so many (eight, in fact) additional commands in the editing language.

It is true that only one directive may be typed on each line, so that it would be necessary to have the two lines

```
DBL
Dl
```

if the DAL directive, for example, were abandoned. This may have been the factor which made the authors decide to provide the eight additional commands. If so, they have not been altogether consistent, since the two directives

```
F/XYZ
F.XYZ
```

which leave the character pointer before the next occurrence of "XYZ" and the line pointer before the next line containing "XYZ", respectively, have no "after" equivalents.

4.b.1.c The user is given no single instruction for replacing one string by another (unless when they are precisely the same length) and the only line replacement command

```
R.string
```

simply replaces the current line with the given string. "Delete" and "Insert" commands will, of course, achieve the same result, but the paucity of "Replace" commands contrasts with the abundance previously mentioned.

4.b.1.d Similarly, there is a special command for deleting a line beginning with a certain string, another for deleting all lines beginning with a certain string, another for checking that the following line begins with a certain string, but none of these has any equivalent for a string that is not necessarily at the start of a line. It may be that these directives - none of them essential - are provided in order to encourage the user

to think in terms of initial strings.

4.b.1.e Although the notation adopted is, for the most part, very clear and helpful, there are some inconsistencies.

Finding a line beginning with a certain character string and finding a line containing a certain string are similar editing procedures (even although the former is economically preferable), yet the COTAN directives are not related, being respectively

```
CB.string      (or simply B.string)
F.string
```

There is no clear reason why the concept of copying should be abandoned in the latter case.

While it is possible to print all the lines being copied by prefixing "P" to a C command, as in

```
PCB.string      (or PB.string)
```

this facility does not seem to be available with the F command.

By typing

```
FP.string
```

the identified line will be printed. The parallel

```
CBP.string
```

is apparently not possible, but the result that could reasonably be expected from that can be achieved by

```
P.string
```

As has been implied in some of these examples, the "C" for "Copy" may be omitted in directives. Although this may save the experienced user a little time, it causes some loss of mnemonic value.

4.b.1.f The editor provides a "LOOP" command, which enables a sequence of directives to be obeyed a specified number of times or until the end of the file is reached.

4.b.1.g The file line is fixed at 80 characters and the normal editing field is from columns 1 to 72. This may be extended to the full 80 characters, if wanted.

4.b.1.h A "tab" facility allows the user to specify the column in which any sequence of characters is to start in an insertion and thus avoid the counting and typing of spaces.

4.b.1.i A useful feature of the system is that the last sequence of directives entered by the terminal user may itself be amended (or erased) if some error has been encountered.

4.c *Ed: the MTS File Editing Program, produced at Newcastle for the Northumbrian Universities Multiple Access Computer.[29]

*Ed may be used on-line or in a batch run.

Editing is performed on a single file, rather than by copying the original to a new file with the necessary amendments introduced. The problem of changes in line numbers does not arise in this case, since MTS line numbering is used and this allows for the insertion and deletion of lines without altering the numbers of existing or remaining lines. Editing usually proceeds sequentially through the file, although it is possible to return to an earlier point by specifying the line number or by giving the "First" command, which makes the first line of the file (not necessarily line no. 1, since negative line numbers are used by MTS) the current one.

Most commands are single letters, although a few are two and one (STR) is three. With the exception of the "String" (STR) command and, in certain circumstances, the "First" (F), "Next" (N), "Reset" (R) commands, all of them may be written in full if the user prefers.

4.c.1 Interesting points

4.c.1.a The string parameters used with certain commands are automatically stored in buffers for possible re-use.

The following "Alter" command, which replaces the first string by the second:

A /ABC/XYZ/

(where '/' is the string delimiting character), will cause the buffer, STR1, to be loaded with "ABC" and buffer, STR2, with "XYZ".

The "Match" command, which searches for a line beginning with a certain string, is associated with the buffer, STR4, so that the command

M /DEFGH/

will place DEFGH in that buffer.

If, by any chance, the next "Alter" command needed the same parameters, the simple command

A

would be sufficient, since the contents of STR1 and of STR2 would be automatically supplied by the program. Similarly, the command

M

would have its parameter supplied from STR4. On the other hand, if the next "Match" command needed the parameter "XYZ", this could be expressed as

M STR2

and the command

A STR4 STR1

would be equivalent to

A /DEFGH/ABC/

There are six string buffers, each associated with a parameter of a particular command, and two spare ones for general use. They are named STR1 to STR8. As well as being loaded indirectly by the use of the associated commands, they may also be loaded directly by the "String" command. If the commands

STR3 /HAND/
STR4 /FOOT/

were given, then the next simple command

M

would be equivalent to

M /FOOT/

and

M STR3

would be equivalent to

M /HAND/

If a "String" command is given without a parameter, as

STR4

this will display the contents of that buffer for checking.

4.c.1.b It is possible in *Ed to limit the field of editing within a line by means of the "Window" command. For editing Fortran programs, for instance, it might be preferable to give the command

W 7 72 or W 1 72

so that there would be no danger of numbering in columns 73 to 80 shifting left after a deletion command, or overflowing after an insertion.

4.c.1.c The number of times an operation has to be repeated can be specified simply by following the command with the appropriate integer. The command

A /ABC/XYZ/ 6

will replace the next six occurrences of "ABC" by "XYZ", while

A /ABC/XYZ/ all

will replace all occurrences of "ABC" by "XYZ" throughout the remainder of the file. "All" is normally abbreviated to "a".

More elaborate loops are possible with the "XEC" command. If the sequence comprises only a few commands, these may be typed on the same line as the "XEC" command:

X (first command; second command; third command ...) [count]

otherwise they are entered on successive lines following the "X" command until the terminating character is read.

4.c.1.d Two commands, the "Copy" and the "Move", enable the user to copy a block of text from one part of the file to another. With the "Move" command, the block is deleted from its original position. These two commands are dependent on the specialised MTS file numbering system.

4.c.1.e A "Sequence" command is provided so that identifying characters and sequential numbering may be put in columns 73 to 80 of the file lines.

4.c.1.f There is a rather unusual "Shift" command which causes the current line to be shifted right or left a specified number of positions.

4.c.1.g The user may return to the previous current line by means of the "Re-set" command. This may not have the effect of cancelling the results of a mistaken command, since the editing is being performed on a single file. If the user were at line no. 23 when he gave the commands

```
D 2
R
```

he would certainly find himself back at line no. 23, but it would (in MTS file numbering) be absent - as would line no. 24. Nine consecutive "Re-set" commands are permitted.

4.c.1.h A number of limited pattern-matching facilities are supplied. With the "Match" command, which searches for a line beginning with a certain string and the "Overlay" command, which replaces characters in the current line with characters from the corresponding position in the command parameter, a "fill" character is used. Wherever it occurs, the corresponding character in the line is left uncompared or unchanged.

The "Blank" command places spaces in a line wherever they occur in its string parameter. (Curiously this contrasts with the "Blank" command of the ICL 1900 #XKYA editor [7] which causes spaces to be inserted in the line in the positions where non-blank characters occur in the parameter.)

Although "Match", "Overlay" and "Blank" operate from the beginning of a line, this may have been set at any required position by a previous "Window" command.

4.c.1.i The useful "Help" command will provide the user with either a complete list of *Ed commands and accompanying general remarks or information about a particular command or term used in *Ed.

4.c.2 Defects

4.c.2.a *Ed suffers from the usual difficulty of editors which try to keep as far as possible to single-letter commands - that of attaching any genuine mnemonic value to them.

After C for "Change", CC for "Conditional change" and CO for "Copy", a K for "Klose" command smacks almost of despair! Then the two parallel commands, which respectively search for a line beginning with or containing a particular string, are given the unrelated names, "Match" and "Scan".

4.c.2.b The only command for deleting lines requires the user to specify the number of lines, starting from the current one. That is,

D 6

would delete six lines from the current one onwards. There may be circumstances in which it would not be easy for the user to determine this number, although he may be able to indicate the end of the deletion quite simply in some other way - by specifying a character string in the last line perhaps.

It is possible to replace single lines by means of the "Alter" or "Change" commands, but to replace more than one line at a time, it is necessary to combine "Delete" and "Insert" directives.

The only "Insert" command is concerned with complete lines. To insert a character string into a line, the "Alter" or "Change" commands would have to be used. For example

C /ABC/ABCDEF/

would insert "DEF" after "ABC" in the current line and

A /DEF/ABCDEF/

would insert "ABC" before the next occurrence of "DEF". This approach presents no particular difficulty, of course, but it may not be immediately obvious to the inexperienced user who is thinking of the operation as insertion rather than replacement.

4.c.2.c There seems to be no satisfactory way for the user to correct errors. As has been mentioned, the "Re-set" command will return to the previous current line, but it will not necessarily undo the effect of the last command or commands. Similarly, the program will cause an automatic return to the previous current line if a search fails, but some changes may have taken place. If the command

A /ABC/XYZ/

has failed and a return has automatically been made, the user may be sure that the rest of the file has not been affected, but if the command were

A /ABC/XYZ/ 6

five replacements may have been made before the final one failed. This may or may not be acceptable to the user.

4.d DOLPHIN: a text filing system for university use; constructed at the University of Lancaster. [30]

The DOLPHIN editor has been included in this survey because it has some interesting features, but unfortunately it has not been possible to obtain any more information about it other than that included in the rather short account in The computer journal of May, 1970, so some details are unclear.

The editor may be used on-line or in batch mode.

Editing is done on a single file and not by creating a new one with amendments introduced. Line numbers are, therefore, not used in addressing.

Editing proceeds sequentially through the file, although some commands reset the file to the beginning.

4.d.1 Interesting points

4.d.1.a There are two editing modes in the DOLPHIN system, the LINEDIT mode, which allows editing to be performed on complete lines and the EDIT mode in which operations are on characters. The first time either of these DOLPHIN "macro-instructions" is given, a copy of the appropriate file is made, ready for the alterations specified in following editing "micro-instructions". Any following EDIT or LINEDIT macro-instruction apparently causes a return to the beginning of the file.

4.d.1.b The chief characteristic of the DOLPHIN editor is its use of pointers. There are 27 of these, "A" to "Z" and a special one named "*".

If the following instruction is given in EDIT mode,

G = "The chief characteristic" = B

the pointer G will have been set at the beginning of the string and the pointer B at the end. Any reference to GB will indicate that segment of text. The instruction

GB = "One interesting feature"

will replace "The chief characteristic" with "One interesting feature". The instruction

GB =

will delete the string.

DOLPHIN is able to identify a string, even when it does not all lie within a single line. If the above instructions had been given in LINEDIT mode, the G pointer would have been set at the beginning of the line in which the string began and B at the end of the line in which it finishes. The instruction

```
GB =
```

would then have deleted the line or lines between the pointers and

```
GB = "One interesting feature"
```

would have replaced the complete line or lines with the specified phrase.

A long character string - or a number of lines - may be identified by setting pointers to the beginning and to the end in two instructions, as

```
K = "The first characters"
"The last characters" = T
```

A particularly useful feature of defining segments of text by pointers is that they may then be very simply interchanged. If two adjacent segments of text have been defined as XY and YZ, they may be interchanged by the simple micro-instruction

```
XZ = YZ + XY
```

A new segment could at the same time be inserted between the others by

```
XZ = YZ + "This is an inserted segment" + XY
```

Similarly if AB and CD have been defined and CD occurs later in the text than AB then

```
AD = CD + BC + AB
```

will interchange AB and CD.

Insertions may be made by using the concept of the null segment. If one defines a segment

A = "Spain" = B

then gives the instructions

AA = "The rain in "
BB = " lies mainly in the plains"

then the former string will replace the null segment AA immediately before the word, "Spain", and the latter will replace the null BB immediately following it.

Pointers may be advanced by specifying an integer increment. Thus

A = A + 4

will advance the pointer A by either four character positions or four lines, depending on the current mode. It is also possible to specify, for example,

A = B + 4

setting the pointer A four characters or lines beyond the pointer B.

4.d.1.c Short strings may conveniently be deleted or replaced without the use of pointers by means of the "equals" sign. The simple statement

"The chief characteristic" = "One interesting feature"

will replace the first string by the second. By omitting the second string, deletion is specified, as in

"The chief characteristic" =

Replacement or deletion may take place throughout the file with the REPEAT command. For example

REPEAT "tulips" = "daffodils"

will replace every occurrence of "tulips" with "daffodils".

The straightforward use of the "equals" sign in nearly all the micro-instructions is one of the greatest strengths of the DOLPHIN editor. It makes an editing program readily understood even by somebody unfamiliar with the system.

4.d.1.d The DOLPHIN system provides the sophisticated facility of justifying lines to contain a specified number of characters. At the end of an editing session the macro-instruction

JUSTIFY, 100

would ensure that the file was rearranged so that each line contained 100 characters - including, of course, the necessary blanks to make the adjustment.

4.d.2 Defects

4.d.2.a There seems to be no way of limiting the "REPEAT" operation to a particular section of the file or to a particular number of occurrences. The file is apparently reset at the beginning when a "REPEAT" command is read.

4.d.2.b There seems to be no way in which a sequence of commands may be repeated a certain number of times or till the end of the file.

4.d.2.c There seems to be no way of cancelling the result of a directive if it turns out to be undesirable - although there is an instruction, "LOSE", which in effect enables the user to cancel all previous commands and start the editing session afresh.

4.d.2.d There seems to be no way for the user to mark the end of his file with a pointer in order to make an insertion there - a common operation - unless he is able to specify it by context.

It should be emphasised, however, that had more information about DOLPHIN been available these disadvantages might have been found to be apparent rather than real.

4.e MITEM: a portable program for the manipulation of text, produced by the U.K.A.E.A. Research Group at the Culham Laboratory in Abingdon.[3]

MITEM is described as "portable" because it is designed for use with different machines. It is available in six versions, increasing in power and complexity from MITEM1 to MITEM6, the facilities of each version being an exact subset of those in any higher-numbered one. Any or all of these versions may be provided by a particular installation, depending on the needs of its users.

Because MITEM is "portable", a number of symbols used in the commands are system-dependent. For convenience, such symbols are represented in the following account of MITEM by the characters used in MITEM: preliminary user manual.

4.e.1 General points

MITEM may be used either on-line or in batch processing.

The editor employs the concept of a pointer moving through each line from character to character as well as from line to line of the file.

Searches and changes are made in the current line or later in the file, not at earlier points.

A new file is created during editing, rather than changes being made to the existing file.

Line numbers (and numbers referring to character positions within a line) are used to a limited extent in a few commands. It is possible to type

C12

in order to set the pointer at character position 12 in the

current line, and

D32,

which would delete from the current line to before line no. 32. It is not possible, however, to delete from line number m to line number n without a preliminary command to set the pointer at line no. m; nor is it possible to use line (or character position) numbers in a "Replace" command.

The commands are mainly single-letter mnemonics, such as C for Copy, D for Delete, I for Insert, J for Jump, but there are exceptions. The "Arithmetic" command, for instance, takes the form (in Backus-Naur notation):

<integer variable>=<integer variable>|
<integer variable><operator><integer variable>

The "Comment" command consists of the non-alphabetic character "/", and causes the rest of the command line to be ignored.

Some related commands are distinguished by a second character, as, for example

V+
V-

the former causing the current line to be printed for verification and the latter preventing the printing. (The "+" and "-" are system dependent and so may vary from one installation to another.)

The authors of MITEM, aware of the limitations of the alphabet, have inserted into the program an escape clause in the form of their "Xtra" command. They describe its function as "to provide extra commands for frequently used operations". In effect, it gives them the possibility of another twenty-six two-letter commands, with X as the first letter. At present only two of

these are in use:

XS
XG

both concerned with the replacement of one string by another.

4.e.2 MITEM1

The simplest version, MITEM1, enables the user

(1) to copy or delete from the current position in the line or file to a required position

(2) to insert a character string at the current position in the line, or lines before the current line of the file,

(3) to reset the pointer to the start of the current line or of the file (in the latter case the remainder of the file being edited will first be copied to the file being created and the pointer will in effect be set to the beginning of the new file),

(4) to set up "tab" positions (to a maximum of 9) for any lines being inserted or expanded.

4.e.3 MITEM6

New facilities are added with each higher-numbered version of MITEM. All these are, of course, available in MITEM6.

4.e.3.a Several features of MITEM6 deserve special mention.

4.e.3.a.1 The edit field may be limited to only part of a line by use of the "Field command. If the original line length were 80 characters, the command

F7 72

would ensure that the first 6 positions and the last eight were unaffected by editing.

4.e.3.a.2 The "End" command limits a search to a particular section of the file. The commands

E56. CXABCX.

ensure that a search for a line containing the string "ABC"

will be made from the current line up to line no. 56, if necessary. The search will finish either when the required line is found or when line no. 56, is reached, whichever is the earlier.

4.e.3.a.3 Ten integer variables (called simply 0 to 9) may be used to hold signed integers and to make simple arithmetic operations possible, with the four operators

\+ - * /

A variable may be filled either from the current position in the line being edited, by means of the "Load" command (which will both read the integer and delete it from the line) or from the command line by the "Hold" command. It is unloaded into the current position in the line with the "Unload" command. If we assume that the pointer is currently set before the characters

56 years

then the following sequence of commands

L5 H7+4 5=5/7 U5

would put the integer 56 into variable 5, the integer 4 into variable 7, replace the contents of variable 5 by the result of 56 divided by 4, then unload variable 5 into the current position in the line. The line would have been changed to

14 years

This example illustrates the use rather than the usefulness of these integer variables. In fact the variables (together with looping facilities not yet mentioned) make possible such operations as the numbering of lines in a file or the altering in some regular way of numbers already there.

4.e.3.a.4 The concept of streams is used to give a considerable degree of flexibility to the editing process.

The following four streams may be considered basic to almost all editors

READ	(text to be edited)
WRITE	(edited text)
CONTROL	(commands)
PRINT	(success and failure messages)

To these MITEM adds another four

MERGE	(text for merging with input)
STORAGE	(duplicate output text)
DELETE	(text deleted from file being edited)
LIST	(program log - includes output to any other stream as well as all commands)

The user is empowered to connect these streams, temporarily or permanently, to numbered channels. He may then, for instance, move a block of text from one position in the file to another by first sending it to the DELETE stream on, say, channel 6, then rewinding channel 6 and connecting it to the READ stream when the point for insertion is reached.

Several files may be combined into one by having them on different channels and connecting these channels as required to the READ stream - or possibly to the MERGE stream, although its use is specialised and therefore limited. Another powerful use of the stream facility is to store on a channel a sequence of commands which can then be connected to the CONTROL stream when - and as often as - needed.

4.e.3.a.5 MITEM employs string buffers to store character strings. The user can avoid having to retype the same character string several times, by placing it in a buffer and then simply referring to the buffer name. A buffer name may be any single character, except for the digits and one or two others, which are reserved for system functions. The buffers are loaded in much the same way as the integer variables by the "Hold" and

"Load" commands. If the character pointer were currently set before the words

THATCHED COTTAGE

then the sequence

LC+8 HDMCOUNTRYM UD

would replace "THATCHED" by "COUNTRY". Buffer C would remain loaded with the 8 characters "THATCHED" and buffer D with "COUNTRY" - the "M" being the string delimiter.

It is not only the "Unload" command that may be used to access the contents of buffers. The buffer name may be used to replace the usual character parameter in most commands. The instruction

C 'D .

would find the next line containing the word "COUNTRY" and

XG 'C TILED

would replace every occurrence of "THATCHED" by "TILED", if we assume that buffers C and D contain the strings previously stored. (The second parameter in an XG and XS command must be an actual character string and not a buffer name. The "prime" character is assumed to have been defined as the character indicating that the following letter is a buffer name. The space is used as a string terminator in the above examples.)

A sequence of commands on one line may be stored in a buffer and called when wanted by the "Obey" command.

The "-" string buffer is used to define characters which are to be ignored in any search for a character string. If the characters "space" and "hyphen" were put in this special buffer then the command

C HANDBOOK .

would find the next occurrence of the word even if it were spelled "HAND-BOOK" or "HAND BOOK". (It would, of course, also find "SECONDHAND BOOKCASE", but the user would presumably be on his guard!)

Much more elaborate facilities for pattern matching, modelled on SNOBOL4, are available through the string buffers. The user has a choice of two methods of availing himself of these facilities, both of which employ symbols for the following functions

OR
TERMINATOR
ARBITRARY
ANY
NOTANY
SPAN

By the first method, he uses the digit string buffers 1 to 6 which have been loaded by the system with the six symbols, in the above order. He may then use the address

XWAGES'1SALARYX

to specify the occurrence of either "WAGES" or "SALARY". The parameter

XAB'3CD'3EFX

specifies a character string beginning with AB, followed by arbitrary characters to CD then more arbitrary characters up to EF.

The zero digit buffer string contains the digits 0 to 9 as a character string, so the parameter

X'6'0'2:X

refers to a character string comprising any or all of the digits and finishing with a colon.

The second method of complicated pattern matching uses not the digit string buffers but the special "*" string buffer. The user may then define his own symbols by loading seven characters into this buffer. The first is interpreted as the string buffer name indicator (shown as "'" in the above examples) and the others are the OR to SPAN symbols in the order given above. If the seven characters are

*.;↑/':

then *A would access string buffer A and the last three examples could be rewritten as

```
XWAGES.SALARYX
XAB↑CD↑EFX
X:*O;:X
```

4.e.3.a.6 A limited but useful conditional facility is available in MITEM with its "Position" command. This command causes the pointer to move to a new position in the READ stream but does not affect the WRITE stream. No failure message is given if it is unsuccessful in its search for a character string, but instead further commands on the same command line are ignored and the next command line requested. If the user wanted to delete all comment lines from a Fortran program, he could use the P command to test whether the letter C appeared in the first position of the current line. If it did, then the next command in the command line (i.e. D+1) would be obeyed, otherwise control would pass to the next command line, which would be the N command to read the next line of the file.

4.e.3.a.7 There is a "Jump" command in MITEM which enables the user to move conditionally or unconditionally to a new command line. This command cannot be used when the CONTROL stream is attached to its usual channel, so programs containing it must be

written to some other channel which is then connected to the CONTROL stream when needed.

The command

J+2.

will cause control to jump two command lines and obey the next. For conditional jumps the values of the integer variables or the contents of string buffers are compared.

Thus the command line

J(3&4):XC+2.X. D+1.

specifies that if the value of integer variable 3 is greater than or equal to that of integer variable 4, then control will be transferred to the command line beginning "C+2.", otherwise the command "D+1." will be obeyed.

The line

J(B=C)+1. C+3.

means that if the character strints in buffers B and C are the same, then control jumps one line, otherwise the command C+3. is read.

4.e.3.a.8 There are several looping facilities provided in MITEM.

The simplest looping function is limited to a sequence of commands which can be typed on one line, with the "Obey" command as the final one on the line. "Obey" in effect causes the preceding commands in the line to be obeyed again. The sequence

C HAND . D+4 O

would cause every occurrence of "HAND" from the current line to the

end of file to be deleted while

```
C HAND .    D+4   O(6)
```

would cause the first 7 occurrences of "HAND" to be deleted.

If the same single-line sequence of commands were needed at different stages in the editing session it could be stored in a string buffer, say buffer G, and called upon when needed by the "Obey" command, as in

```
OG
OG(7)
```

When the desired sequence for a loop cannot be typed on a single line, the sequence of commands must be written to some channel other than the main CONTROL one, and that channel connected temporarily to the CONTROL stream when needed. The "Obey" command may again be used for this purpose. If a sequence of ten command lines has been written in channel 9, then the directive

```
O9
```

will cause that sequence to be obeyed once. If the directive is

```
O9(50)
```

then the sequence will be obeyed 5 times, since the integer in brackets specifies, in this case, the number of lines to be obeyed from channel 9.

Even without the "Obey" command, loops may be used in MITEM by the facility for rewinding channels and attaching them to streams. If, again, the user wanted to remove all comment lines from a Fortran program, he could use the following sequence:

```
I(9<)ZZ
E;.  P:XCX   D+1.  SC<
N   SC<
Z
SC9<
```

This specifies that the second and third lines above should be inserted on channel 9, rewound. Then the "Stream" command SC9< rewinds channel 9 and makes it the temporary CONTROL stream, so that the commands on it are obeyed. The "Position" command tests the first character in the current line. If it is C, then the line is deleted and the "Stream" command rewinds the current CONTROL stream (i.e. channel 9) and the first commands on it are again obeyed. If the first character of any line is not C, then the remaining commands on the control line are ignored, and the next control line is obeyed - that is, the "Next" command causes the next line of the file to be read, the current CONTROL stream is again rewound and its first command line re-read. The looping continues until the end of file is reached when the "End" command disconnects channel 9 and reconnects the normal channel to the CONTROL stream.

The same looping effect as in the above example could be achieved by using the "Jump" command instead of the "Position" one. If "C" were stored in one string buffer and the first character of each line loaded in its turn into another, the two buffers could be compared each time.

4.e.3.a.9 The "Comment" command, which causes the rest of the command line to be ignored, could be particularly useful in off-line editing or when a file of commands was being built up for later use.

4.e.3.b Defects

MITEM has clearly - at least in its higher-numbered versions - great power from its concept of streams and channels, from its string buffers and pattern-matching facilities and from its conditional testing.

The faults that may be found with it lie rather in its use for simple editing, where the user is not interested in its sophisticated techniques nor in its underlying theory but only in the practical service it can give him,

4.e.3.b.1 One very minor difficulty - common to most editors that employ the concept of a character pointer - is that it is only too easy to type a command referring to file lines in mistake for one referring to characters within a line (or vice versa) since the only thing that distinguishes them is a symbol at the end of a line command. For example

```
DXTIMEX
DXTIMEX.
```

mean, respectively, "Delete from the current position in the

line to the end of the character string, 'TIME', in the line" and "Delete from the current line to before the line containing 'TIME'." It may reasonably be argued, however, that the advantages of having the line and character commands the same, where possible, outweigh the danger of unintentionally omitting the terminating symbol of a line command.

4.e.3.b.2 A much more serious source of danger and confusion is the MITEM concept of the "changed line" as it affects the subsequent search for a new current line. A number of commands (such as C, D, P) may make use of an increment address to change the current line, thus

P+3.

will move the pointer forward three - or it may be four! - lines. If the current line has not had any changes made to it, then the number of lines will be three; if it has been changed, then the pointer will move four lines. Although it is easy to sympathise with the anxiety of the MITEM authors to preserve any changes that have been made to a line, it seems a quite unnecessary complication for the user that "D+1." should sometimes delete the current line and sometimes the line after the current one. An added difficulty is that the interpretation of "changed" is that of the program and not of the user. If the user, during an on-line editing session, gives the following sequence of commands

```
C+4.    CXBIRDX
D+2.
```

(we may assume that he had second thoughts after accessing the word "BIRD" in the current line!) he may well find that he has not deleted the intended two lines. By giving the command

"CXBIRDX" he has "changed" the line from MITEM's point of view, even although from his it is unaltered.

4.e.3.b.3 MITEM makes use of two symbols, one indicating the last character in a line or the last line of a file and the other the character or line after the last. Although it is sometimes convenient to have such symbols available, an inconsistency is introduced in that the last character in a line is initially taken to be the last non-blank character, whereas if the user has changed the line and introduced some trailing blanks, then the "last character" symbol will access the final blank. The command

C;

may mean "Copy to after the last non-blank character" or "Copy right to the end of the line" depending on the preceding commands. A rather similar complication is introduced with the "Field" command, where trailing blanks are ignored, so that an attempt to access the last character position by, say, "XTE X", will fail, since the blank will have been, in effect, removed.

4.e.3.b.4 The user with fairly modest editing demands may find that, while all his needs are certainly met, he may have to use two or even three directives to accomplish an operation which could be done with one in some other editing systems. As has been mentioned previously, in order to delete lines 26 to 35, he would first have to make line 26 the current one, that is, it would be necessary to type

C26. D36. or C26. D+10.

rather than

D26.-35.

Although there is a "Replace" command which will achieve a line-for-line replacement, if the number of new lines is not the same as the number of old ones then resort has to be made to a combination of "Insert" and "Delete" commands.

4.e.3.b.5 Such restrictions are not, of course, serious and are presumably due to the desire to keep as far as possible to single-letter directives, but they do contrast markedly with the almost over-generous choices provided in other ways. There are, for instance, two increments which may be used in addresses, the CUR increment (written as "+" in the preceding examples) and the AFT increment (which will be shown as "*"), which indicates that counting has to begin from the next character position or line, so that

```
C+1 ≡ C*
C+3. ≡ C*2.
DXWAYX+4. ≡ DXWAYX*3.
```

The only difference is that with a character address, counting with the CUR increment begins at the first character of the string while with the AFT it begins at the last character. The execution of the command

```
CXWAYX+2
```

leaves the pointer before the "Y", whereas

```
CXWAYX*2
```

leaves it three positions beyond. Since the user may easily achieve the same result by putting

```
CXWAYX+5
```

the AFT increment seems superfluous.

4.e.3.b.6 The same is not quite true of either of the methods of calling the functions used in defining complex patterns discussed in Section 4.e.3.a.5 and yet might it not be possible

even here to have only one method? The advantages of both methods could be combined, either by allowing the user to alter the contents of the digit string buffers if the pre-loaded characters were unsuitable or by pre-loading the "*" buffer so that the user need change it only if necessary.

4.e.3.b.7 It could also be argued that there is no need for both the "end of line/file" character and the "position-after-end of line/file" character, referred to in Section 4.e.3.b.3.

4.e.3.b.8 The "Field" command, which limits the edit field to part of the line, has one disconcerting side-effect in that it also causes the current line to be reread. This could mean that the unsuspecting user, having made changes to the current line and then given the "Field" command in preparation for the next editing instructions, would not only lose the changes he had made but might also find himself with an incorrect new current line if he made allowances for a "changed" line, when in fact there had not been one! It should be added that this command may also be used deliberately to cancel any mistaken alterations to the current line, but it would seem more satisfactory to have a separate command for this purpose, if it were felt to be important, rather than to leave the action of this one as it is - something of a trap for the unwary.

4.e.3.b.9 There is no very simple way for the on-line user to cancel the effect of a faulty command and return to his previous position for another attempt. It is true that if a search fails the program will automatically make this return (unless the "End" command has been used), but the problem remains in the cases where the editor cannot detect the error. If the user is quite sure of his former positions in both the READ and WRITE streams,

he should manage to recover his position by rewinding or back-spacing these streams, but otherwise he has no real escape.

4.e.3.b.10 With its (chiefly) single-letter directives, its use of a large number of non-alphanumeric characters for special purposes, and its sensible but disconcerting use of the digits as integer variable names, the language of MITEM is not readily memorised. The following sample program is taken from the Preliminary user manual:

```
I(9<)ZZ
C:Z  ('6'OZ/Z(Z*   L1  J(1≠2)+.  SC3
J(3=1)+.  1=1+4
U1   SC<
Z

H2+121  H3+39  H4+5
O9
C;.
```

This is clearly not very easy to read or to interpret. On the other hand, it is accomplishing a fairly complex operation (incrementing by 5, numbers from 40 to 120 occurring in a certain position in some lines) which few other program editors could achieve - at least in such a concise form.

CHAPTER 5

DESCRIPTION OF THE PRESENT EDITING PROGRAM

The general features of the present editing program are described first; then an account is given of each command, explaining its action and the reason for its inclusion in the editor. A list of error and information messages is provided at the end of the chapter.

5.a General design features

As stated in Section 1.e this is a program editor and is unlikely to be suitable for "free-form" text. It includes no formatting facilities. A sought-after character string will not be identified if it is divided between two lines.

Before actual editing begins, a copy of the file to be edited is made. Editing is carried out on this copy, so that the original is unchanged. As a by-product of copying, the number of lines in the file is established. This information can be used to detect, at an early stage, certain errors in certain commands.

A new file is created during editing. As indicated in Section 3.b, this makes deletion and insertion of lines very simple,

allows the use of line numbers for addressing, makes it possible to include means of recovery from on-line editing errors - and makes sequential editing advisable. A few special commands involve backward movement of the files, but generally all movement is forward.

Although line-number (as well as context) addressing is supplied, there is no facility within the editor to produce a file-listing with line numbers, since RAX has an on-line command (/DISPLAY) which provides such a listing.

A line pointer is, of course, maintained and the concept of the current line will be frequently employed throughout this description. No character pointer within the line is used (see Section 3.d) which means that each command operating within a line contains a context parameter for addressing.

No notional line, either before the first line of the file or after the last, is used. This being so, there have to be both "insert before" and "insert after" facilities, to allow insertions at the beginning and at the end of the file. Similarly, both operations have to be supplied for use within lines.

Insertion of characters in a line involves a loss of space characters at the end of the line. Blanks at the start of a line can be lost only by a specific deletion operation - they cannot be "pushed off" at the left by an insertion. If an insertion would result in non-blank characters overflowing, these are not lost but are preserved in an extra following line - or lines. This is not usual in program editors. More often the surplus characters are lost and an error message printed. It has been thought better, however,

to assume that the user knows what he wants, until it is proved otherwise. If he has, in fact, misjudged the length of an insertion, then he has to correct his mistake, whichever system is chosen.

Because the editor is designed primarily for interactive use, the commands are obeyed interpretively. If one command fails, the program attempts to execute the next (but the user may change this normal procedure). No preliminary sorting of commands is possible, even in background editing (this would in any case be feasible only if all commands included line numbers) and an error message is given if a command is read referring to a line number lower than that of the current line.

The user may ask for the current line (and the following ten lines, if wanted) to be displayed or printed. He may ask also for the display or printing of all new (amended or inserted) lines, or of all lines as they are being written to the new file, in order to check the progress of the editing.

The file being edited may be reset to the beginning at any stage. This permits such operations as the interchange (albeit clumsy) of blocks of lines of the file.

It is possible to reset both files to the beginning and start again if something has gone badly wrong - or to cancel the results of the last few commands, if the error is not so serious. Obviously this is useful only in on-line editing.

The length of line (assumed to be 80 characters) may be altered by the user before actual editing begins. The maximum permitted length is 132 characters.

The units from which the file to be edited, the commands and the additional lines for insertion are read and to which the edited file is written may be changed by the user. Although this is primarily intended for use before editing, in cases where the default options are unsatisfactory, the sophisticated user may achieve some manipulation of files by changing some of these units during editing.

Fairly detailed error and information messages are given, in the hope that the user may easily find the cause of any mistake.

5.a.1 Format of the commands

The more frequently used commands vary from one character to four characters in length. Each has been kept as short as is compatible with the need to distinguish it from others and to provide it with some mnemonic quality. The first character is always a letter, since it is thought that consistency in this respect is less confusing for the user than to use some initial non-alphabetic characters - such as the "-", "+", and "*" of the Cambridge EDIT.[17,39] Similarly, a command is always stated and not merely implied, as in some editors (see Section 3.f).

"D" is used as the first letter for all deletion commands, "R" for all replacement ones, "P" for all printing commands and "F" for all commands which involve finding a particular line. "C" for "Copy" might well have been chosen (as in several other editors) instead of "F". The only advantage the latter has is that it may convey better the idea that the found line is the relevant, current one, whereas "Copy" may suggest that the particular line has actually been written to the new file.

As far as possible, the second or third characters have been used with the same meanings in the four main groups (D, I, R, F), but this could not always be achieved. In insertion commands, for example, "B" implies "before" as against "A" for "after", while in D and F commands it stands for "beginning" as against "C" for "containing". It is hoped, however, that no confusion will be caused by this in practice. The last line of the file is referred to by the letter "L", in the FL and DL commands. This agrees with several other editors, including the COTAN Amender [31] and MTS *Ed. [29] Others use a special character (the dollar sign of QED [14] and the asterisk of KOS EDITDC [32]) but there seems no advantage in this.

As we shall see, some of the less used commands are longer than four characters and some are represented as complete words. This is partly because satisfactory abbreviations would have been hard to find, partly because brevity is not so important when the commands are not frequently typed, but chiefly to emphasise that they are different in character from other commands.

When the editor was written, the RAX conversational read facility would accept only one line of input at a time. To overcome this disadvantage, several commands may be typed on one line, with the semi-colon as a separator. Reading several commands at once, whether they are all on one line or one to a line on several lines of the screen (as is now possible) may lead to a serious accumulation of errors, if a mistake is made in an early command, so a special command (EXIT) is provided to give the user some control over this.

The ability to put more than one command on a line has made it easier to limit the number of commands. The COTAN Amender, which accepts directives only one to a line, has found it necessary to include such pairs of directives as

DAL
DBL } Delete to after/before last line

In the present editor, "DL" is equivalent to the second COTAN instruction, while the first COTAN one can be expressed by the addition of the "Delete current line" instruction:

DL;D

As few demands as possible are made on the user in the typing of the commands. Many editors require that each command be given on a new line, start in column 1 and contain no blanks. This makes implementation more efficient. Once it is decided, however, to allow several commands on each line, efficiency is already lost and a fixed format more difficult to impose.

The user may insert spaces where these seem most natural to him; the editor takes account of them only within character string parameters. There blanks are treated as ordinary characters - except at the beginning of the parameter in a DB or FB command (where the string is specified as occurring at the start of the line). This exception is made to avoid the necessity of counting and typing initial blanks - and to avoid the errors that this would entail.

This practice of taking blanks into consideration, even in identifying strings, may be rather different from the usual one in editing languages. Certainly those that make their policy clear - KOS EDITDC and the COTAN Amender, for instance - ignore blanks when

identifying strings.

It is not easy to decide which approach serves the user better. It would no doubt be annoying for him, if he were trying to reach the line

GO TO 6

to find that it had not been identified because in the original program it happened to have been punched as

GOTO 6

It would, on the other hand, be equally annoying if he were trying to delete all occurrences of "always" and found that "the ideal wayside inn" had been transformed into "the ideide (or possibly 'ideaide') inn".

Since it is clearly impossible to avoid all misunderstandings in this regard and since blanks must be considered where deletions or insertions are involved, it has been decided to consider them also in addressing - with the important exception in DB and FB commands, mentioned above.

As there is no fixed format for the typing of commands, blanks cannot be used as delimiters of string parameters. Delimiters may be any other character - except E or S - not occurring in the particular strings. It is, in fact, only with the DS command that S is an unsuitable delimiter and with DS and RS that E is unacceptable (because of confusion with the DSS, DSE and RSE commands), but it seems preferable to rule them out altogether rather than to explain more precisely to the ordinary user when they should be avoided.

A command, with any parameters, must all be typed on one line and not split between the end of one line and the beginning of the next - unless it is one with very long string parameters which cannot be contained on one line. In this case, a continuation character must be typed in column 1 of the second (and third, conceivably!) line. A string parameter may be as long as the text line length, although it is difficult to imagine that the user will want to make it so.

Lines for insertion after an IA, IB or R command and material following a LOOP command must start on the line following the command, rather than continue on the same line. The terminating character needed in each of the four cases must be in column 1 on a line by itself. These requirements are discussed further in Section 5.b.4.a.

5.b The editing commands

In the following account of editing operations and commands, "m" and "n" represent unsigned, positive integers. (The editor would fail to recognise the command if a sign were used.)

Character string parameters are referred to as "string", "string1" or "string2".

The solidus (/) represents the character string delimiter (which could, in fact, be any character, other than blank or E or S, not contained in the relevant strings).

The variable "char" represents any single character other than a blank or a semi-colon.

5.b.1 FIND commands (see Example 1)

The FIND commands search for the specified line and make it the current one. All lines from the previous current one up to, but not including, the found one, are written to the new file in the process. If the line identified by an F command is already current, no message is printed and editing continues with the next command.

The commands are:

F n — Find line no. n.

If n < no. of current line or > no. of lines in file, an error message (no. 15) is printed and no action taken.

FL — Find last line.

For on-line use, this command is not essential, since the number of the last line may easily be established by use of the P command (Section 5.b.8). FL, however, is simpler and may be needed in background jobs, if no file-listing by line number is available.

FB /string/ — Find next line beginning with "string".

FC /string/ — Find next line containing "string".

If the line specified in an FB or FC command is not found, the file will have been read and copied up to the last line, which will be current. An error message (no. 12) is printed. No automatic return to the previous position in the file is made after the failure of a command, since this may result in even greater confusion when attempts are made to execute later commands. The RESET command

(Section 5.b.7.b) provides a means of rescue for the interactive user after such a failure.

5.b.2 MOVE command (see Example 1)

The MOVE command advances a specified number of lines beyond the current one.

The format is:

M[n] Move n lines beyond the current one.
If n is omitted, it will be assumed to be 1.

In response to the command, the current line and the following n-1 lines are written to the new file. The new current line is no. "no. of current line + n".

If "no. of current line + n - 1" is equal to the number of lines in the file, the end of file has been reached (that is, there is no new current line) and an information message (no. 4) is given.

If "no. of current line + n - 1" is greater than the number of lines in the file, an error message (no. 15) is printed and no attempt is made to execute the command.

If n is zero, an error message (no. 8) is given, since such a command would have no point.

An argument could be made for "F + n" for this directive, in line with "D + n" and "R + n" (Sections 5.b.3.a.1 and 5.b.5.a.). It is believed that moving on may be a more natural way for the user to think of the process.

5.b.3 DELETE commands

5.b.3.a Deletion of complete lines (see Example 2)

Deletion is performed by not writing the specified lines to the new file.

5.b.3.a.1 Commands with integer parameters

Although four commands with integer parameters are provided, none is precisely analogous to the F n command.

If the execution of any of these commands involves the deletion of the last line of the file, a message (no. 4) is printed to inform the user that the end of file has been reached. The commands are:

D[+ n] Delete current line [and following n lines]. The new current line is no. "no. of current line + n + 1". If, however, "no. of current line + n +1" is greater than number of lines in the file, an error message (no. 15) is printed and no action taken.

After the execution of any of the following three commands, line no. n + 1 will be current. If m or n is less than the number of the current line or if n is greater than the number of lines in the file, an error message (no. 15) is printed and no action taken.

DA n Delete from (i.e. including) current line to after line no. n.

D n Delete line no. n.

D m/n Delete from line no. m to line no. n, inclusive. If m is greater than n, an error message (no. 14) is printed and no action taken. If m equals n, the line is deleted.

It is not, of course, essential to have all four fommands, since "D m/n" could be replaced by "F m; DA n" and "D n" by "F n; D". There seems no point, however, in withholding from the user the simplest methods of performing some very common operations - but it is appreciated that this argument is a subjective one and may appear to be contradicted elsewhere!

5.b.3.a.2 Three further commands delete complete lines. They are:

DL	Delete from (i.e. including) current line to before last line of file.
DB /string/	Delete from current line to before the next line beginning with "string".
DC /string/	Delete from current line to before next line containing "string".

These three commands are analogous to the corresponding F ones and the specified line becomes the new current one in each case. If the specified line is already current, an error message (no. 11 or 15) is printed. This is different from the reaction to an F command in a similar case. Whereas the user may give an F command just to confirm his position, it is likely that he will believe there is deletion to be done when he issues a D command. It may be claimed that the F command has been successful, since the line has been identified, while the D command has failed, since no deletion has been performed.

If the line specified in a DB or DC command is not found, the file will have been read but not copied up to the last line, which will now be current. An error message (no. 13) is printed. As with FB and FC, no automatic return to the previous file position is made.

5.b.3.b Deletion within a line (see Example 1)

The available commands are:

DS /string/ Delete first occurrence of "string" in current line.

DSE /string/ Delete every occurrence of "string" in current line.

This avoids the need to type several DS commands in certain circumstances. It may help to simplify the problem of repeating corrections throughout a section of the file. This point will be discussed in more detail in Section 5.b.10.e along with the LOOP command.

DSS /string1/string2/ Delete from first occurrence of "string1" to first following occurrence of "string2", inclusive, in current line.

This is not an essential command, but saves the user from having to type a long string in the DS command. If the string is really long, the whole line could, of course, be deleted and the new shorter one inserted. The DSS command may well provide a simpler solution in many cases.

In the execution of these instructions, characters to the right of "string" are shifted left n positions, where n equals the number of characters in "string", and n blanks are inserted at the end of the line. (For DSS, "string" = first character of "string1" to last character of "string2".)

If all non-blank characters in a line are removed by one of these commands, the line is assumed to be deleted. This assumption is open to attack on the grounds that the user may want a blank line and that if he had intended to delete it, he could have done so by the simpler command, D.

If a given string cannot be found in the current line, an error message (no.10) is printed and no action taken.

5.b.4 INSERT commands

5.b.4.a Insertion of complete lines (see Example 3)

The commands are:

IA[n] Insert the line(s) following the command after the current line [after line no. n].

The new current line is the one after the previous current line [line no. n + 1] - unless the previous current line [line no. n] was the last line of the file, in which case a message (no. 4) will inform the user that the end of file has been reached.

IB [n] Insert line(s) following the command before the current line [before line no. n].

The current line is unchanged [is line no. n].

With either command, if n is less than the number of the current line or greater than the number of lines in the file, an error message (no. 15) is printed and no action taken.

As already mentioned (Section 5.a), both "before" and "after" instructions are necessary.

It is not essential to include a line-number addressing facility, since, for example, "F n; IA" could serve for "IA n". If line-numbers are included, why not "Insert after line beginning with 'string'" or "Insert before line containing 'string'"? There is no very satisfactory, objective answer. It can only be repeated that line-number addressing is so simple that it seems a pity not to include it, while to add context-addressing facilities as well would make the editing language unacceptably clumsy.

The requirement that the first line for insertion must follow on the line after the command and not on the same line (as is allowed in some editors) is to avoid any difficulty over the intended format of the inserted line.

The requirement that a terminating character must follow the line(s) for insertion, in column 1 on a line by itself, is rather out of keeping with the fairly free-form typing of commands usually permitted by the editor! It is, however, the simplest and most economical way - and one likely to be familiar to most users - of performing the required operation. Some editing programs regard such terminators as commands in their own right (cf. COTAN "ENDINSERT" and "ENDLOOP").

5.b.4.b Insertion within a line (see Example 1)

The available commands are:

ISA /string1/string2/	Insert "string2" after the first occurrence of "string1" in the current line.

ISAE /string1/string2/ Insert "string2" after every occurrence of "string1" in the current line.

ISB /string1/string2/ Insert "string2" before the first occurrence of "string1" in the current line.

ISBE /string1/string2/ Insert "string2" before every occurrence of "string1" in the current line.

Both "before" and "after" commands are needed (see Section 5.a) and the "every" ones may eliminate some typing. They will be referred to again in Section 5.b.10.e, along with the LOOP command.

If "string1" cannot be found in the current line, an error message (no. 10) is printed and no action taken.

As mentioned in Section 5.a, no error message is given if non-blank characters overflow at the end of a line and these surplus characters are used to form a new line - or new lines. The editor sets no limit on the number of such lines (presumably the maximum possible would be eighty, if the text line were eighty characters!) but it is unlikely in practice that many would be created in this way, when the user is aware that the editor provides no formatting facilities. Since only one line can be in memory at once, the last new line that results from an overflow is treated as the current line, with the same line number as the original, unexpanded line.

5.b.5 REPLACE commands

There is no real need to include replacement operations in an editor, since these can be performed by insertion and deletion. Alternatively, insertion and deletion may be treated as forms of replacement with one null parameter. This editor, in common with most others, employs all three operations, so giving the user the chance to select the process that seems simplest and most appropriate to him in the circumstances - or, more accurately, in some circumstances. If he wants to replace a section of his file from the line beginning with "ABC" to the line beginning with "XYZ" and does not know the line numbers, he will find no replacement command to help him. Instead he will have to use a sequence of commands, such as

FB /ABC/;DB/XYZ/; D; IB

5.b.5.a Replacement of complete lines (see Example 3)

The four commands, chosen as the most straightforward and likeliest to be used are:

R[+ n]	Replace the current line [and following n lines] with line(s) following the command.
R =	Replace an equal number of lines, from the current one onwards, with the line(s) following the command.
R n	Replace line no. n with line(s) following the command.
R m/n	Replace lines no. m to n inclusive with line(s) following the command.

Error and information messages connected with the R [+ n], R n and R m/n commands are the same as for the corresponding D commands (see Section 5.b.3.a.1). Again R n and R m/n are simple and useful, but not essential.

The format of the R= command is perhaps not very pleasing, with the possible interpretation that something is being replaced by the null string - or simply that something has been forgotten! "RE", with the "E" standing for "every" or "equal", might have been adopted, but is not clearly better.

Whatever the format, the command is needed, since there is no alternative means of achieving the same result. It is expected that the command will most often be used when there is a large replacement to be made, possibly with the inserted lines being read from a disk or tape file, rather than typed at an on-line console.

If the last line of the file has been replaced and some lines remain to be inserted, an error message (no. 18) is printed. This includes - for information - the first replacement line which could not be accepted.

These replacement commands have the same format requirements for the typing of replacement lines and terminating character as the IA and IB commands (see Section 5.b.4.a).

5.b.5.b Replacement within a line (see Example 1)

The commands are:

RS /string1/string2/	Replace the first occurrence of "string1" in the current line by "string2".

RSE /string1/string2/ Replace every occurrence of "string1" in the current line by "string2".

If "string2" is shorter than "string1", then replacing is equivalent to overwriting and deleting, with the same effect on the structure of the line as in the DS commands (see Section 5.b.3.b).

If "string2" is longer than "string1", the operation is equivalent to overwriting and inserting, with the same effect on the structure of the line as in the IS commands (see Section 5.b.4.b).

If "string1" cannot be found in the current line, an error message (no. 10) is printed and no action taken.

5.b.6 CONTROL commands

There are five commands which may be thought of as exercising some general control over an editing session: BEGIN, END, RESTART, EXIT, and RETURN. These are not abbreviated, but typed as full words (see Section 5.a.1).

5.b.6.a BEGIN command

This must be given as the first command of an editing session. It will not be recognised elsewhere.

It is needed to technical reasons.

The normal procedure of the editor is to read and copy the file to be edited (assumed to be a disk or tape file) and then to read the commands - from the console, for on-line editing and from the card-reader for background.

If the user wants to exercise any of the options open to him which affect the reading of the file or of the commands (see Section 5.b.9), then this information must be given to the editor before it enters its normal procedure.

The only satisfactory way to do this is to require the user to give an initial command line, which will make clear whether or not the text line length is altered and whether the file for editing and the commands are to be read from the expected devices.

If the default options of the program are satisfactory, then BEGIN will be typed on a line by itself. If changes are required, then the appropriate option commands must follow on the same line as BEGIN, (see Examples 1 and 8).

For on-line editing, the BEGIN line must, of course, be typed at the console (even although the rest of the commands may be read from a previously stored file) and off-line, it must be the first card read by the card-reader.

5.b.6.b END command

This is given as the last command of an editing session.

Any lines not yet copied to the new file are first written to it and the new file is then sent to the output device.

5.b.6.c RESTART command (see Examples 1B, 4 and 7B)

This makes the new file into the file to be edited. Any lines not yet copied to the new file are first written to it and then its first line is the new current one.

This command is useful if any errors or omissions have been made during the first editing pass through the file, or if repeating changes are necessary and cannot be satisfactorily accomplished in a LOOP sequence (Section 5.b.10).

It is a fairly common facility in editors, being equivalent to the COTAN Amender "START" and the Cambridge EDIT "*", for instance.

5.b.6.d EXIT command (see Examples 7A and 7B)

This command may be considered as setting a switch or flag or controlling a mode. After it has been given, an error causes the following reaction:

(a) in interactive editing, any further commands that have been read are ignored and the program awaits another command from the user. The EXIT command is no longer in force once control is returned to the user, so it has to be given again, if required.

(b) in background editing, any further commands are ignored and an END command (Section 5.b.6.b.) assumed.

In either case, an error message (no. 6) is given.

Whether to attempt to execute later commands after an error has been found and perhaps thereby cause an accumulation of errors, or to stop editing immediately, when perhaps all later commands could have been executed successfully, is a difficult decision to make in creating an editing program.

The provision of this command means that one possibility (to try to continue) can be built into the editor and the other also made available to the user.

In interactive editing, if commands are read one at a time, there is, of course, no problem, but if several are read at once, then the EXIT command may be helpful. It is, however, chiefly of benefit in background use.

5.b.6.e RETURN command (see Example 6)

This returns the file being edited to the beginning, leaving the amended file at the point it has reached.

The first line of the file being edited is again current.

The purpose of the command is to allow some manipulation of blocks of lines of the file - the interchange of two sections, perhaps.

The line that is current when the RETURN command is given is not copied to the new file before the return is made.

5.b.7 CANCEL commands

The two instructions for cancelling the effects of earlier commands - ERASE and RESET - could be regarded as part of the CONTROL command group, but it is convenient to consider them separately.

5.b.7.a ERASE command (see Examples 1A and 1B)

The action of this command is to reset both the file being edited and the amended file to the beginning, so that, in effect, all previous amendments are cancelled. The first line of the file being edited is again current.

This is a means of starting afresh in interactive editing when things have gone badly wrong. It should be noted that the effect of a RESTART command (Section 5.b.6.c) cannot be undone and that the switches set by EXIT (Section 5.b.6.d) and some of the print commands (Section 5.b.8) remain unaltered by ERASE.

5.b.7.b RESET command (see Examples 1C - F)

The format of this command is

RESET [n] n may be 1, 2 or 3. If it is omitted, it is assumed to be 1.

Reset both files to the position they were in before the last (or second last or third last) time either of them moved.

With this command, the interactive user may cancel some of his recent amendments.

There seems little point in providing a facility for cancelling changes within the current line. If anything goes slightly wrong there, it can be corrected by another DS, IS or RS command. If there is something seriously wrong, it is fairly easy to replace the whole line with the correct one.

Because of this and because it is meaningless to suggest cancelling the effect of certain commands (particularly the P command for displaying the current line), it is not possible simply to give RESET the function of cancelling the last one, two or three commands.

The idea of returning to the position of the files before the last (second last, third last) movement is a simple one, but there are difficulties in explaining it simply to the user, who may not want to concern himself too minutely with the implementation of the editor. It would be clearer to take account of the movement of the original file only, but this would mean that the result of an IB command could not be readily cancelled, nor, if an extra line were inadvertently created by an IS or RS command, could the mistake be easily corrected, as, in either case, only the new file has advanced.

If only one command could be read at a time, then RESET, without a parameter, would be adequate. Since a whole lineful of commands may be entered at once, the user is given the chance to make up to three backward jumps. This is believed to give him reasonable room for manoeuvre without increasing unduly the possibility that he may miscalculate.

Any changes previously made to the line that is again current after the execution of a RESET command will have been lost.

Successive RESET commands cannot be given, whether with the object of moving farther and farther back in the file or in order to cancel the action of the preceding RESETs. Similarly, the use of RESET is not allowed to cancel a control operation - ERASE, RESTART, RETURN - that involves the rewinding of one or both files.

If any of these operations is attempted or if the parameter of RESET is greater than the number of backward jumps possible at the time, an error message (no. 22) is given and no action taken.

If n is less than 1 or greater than 3, an error message (no. 8) is given and no action taken.

5.b.8 PRINT commands (see Examples)

The main printing command takes the form:

P [+] Print the current line [and the following 10 lines], together with the number of the current line and the total number of lines in the file.

If the editor had been solely for use with the 2260 terminal, eleven lines could always have been displayed and only one form of the command would have been necessary. The "current line only" form has been provided chiefly for the teletype user, but also for the background user, who may find it helpful for checking the progress of editing if something has gone wrong.

The statement of the number of the current line and of the total number of lines is to encourage the use of line-number addressing, even when no line-numbered file-listing is available. For instance, if the user wants to reach a point four lines before the end of the file, he is enabled to calculate its number very easily.

To avoid frequent repetition of the P command, a flag could have been used to cause the printing of the current line after the execution of each command (much as is done by the "Verify" flag of MTS *ED and the Cambridge EDIT). It is, however, so simple to attach ";P" to any command that the flag has not been used.

The dangling plus sign of the P + format may have the same incomplete appearance as the R = command, but it seems as satisfactory as any other solution. There is little point in extending the choice with a P + n command (although this might sometimes have been helpful for the teletype user) and no point at all in requiring the user to type "1O" when there is no choice.

Flags are set "on" and "off" by the remaining PRINT commands:

PNEW	Print all amended and inserted lines.
PRINT	Print all lines as they are being written to the new file.
PCOMM	Print each command as it is executed.
XPNEW	Cancel the previous PNEW command.
XPRINT	Cancel the previous PRINT command.
XPCOMM	Cancel the previous PCOMM command.

The lines printed when the first three commands are in force are preceded by "NNN", "FFF" and "CCC" respectively. (The system messages of the editor have the prefix "XXX".)

The purpose of setting the flags to "on" is to enable the user to check the progress of the editing.

The commands PNEW and XPNEW are taken from the COTAN Amender (where they are apparently used only for inserted lines) and the format of the others modelled on them.

5.b.9 OPTION commands

The commands in this group are for use when the default options of the editor are unsatisfactory.

LINE n (see Example 1)	Treat each line of text as n characters long. ($0 < n \leq 132$. Default option 80)

Within a LOOP sequence (Section 5.b.10), lines for insertion in the text are read initially as command lines, with a fixed length of 80 characters.

If n is zero, error message no. 8 is given; if n is greater than 132, the message is no. 23. In either case the editing session is brought to an end, because the editor is unable to take its next step - the reading of the input file.

UI n	Read input file for editing from unit n.
UO n	Write edited output on unit n.
UC n	Read commands from unit n.
UA n	Read added lines (in replacement and insertion commands) from unit n. This does not affect the reading of added lines within a LOOP sequence (Section 5.b.10), which is always from the unit for commands.

The possible values for n in the last four commands depend on whether the editor is being used under RAX or under 44MFT. This matter is discussed more fully in the User's Guide, Appendix A.

While the above facilities allow the user a little latitude, they are not sophisticated. For instance, although it would be possible during editing to switch to another unit in order to read a sequence of previously-stored commands and then to switch back to the original stream, there is no means of rewinding and re-reading that sequence, so no programming capability exists. Also, although files on different units could be inserted in the main file by the UA command, the material would have to be prepared for the use of the editor - that is, it would have to have the terminator for insertion at the appropriate points.

Where necessary, information about the units used by the editing program is given in the User's guide. With the help of this information, the user is expected to choose a satisfactory value for n in a U command. The editor issues no error messages and will obey the commands (even when potentially disastrous), unless prevented by the operating system.

In the following summary of the commands concerned with terminators, "char" may be any single character other than a blank or a semi-colon.

TA char — Use "char" as the terminating character for a sequence of added lines. (Default option £) (see Example 3)

TL char — Use "char" as the terminating character for a LOOP sequence. (Default option *) (see Example 4)

In both the above cases, the need for the command would arise if a line in the sequence - intended for insertion - happened to have the relevant character in column 1.

TI char — Use "char" as the terminating character for the input file. (Default option £) (see Example 8)

Since the user may choose to read the input file for editing and the file of commands (or part thereof) and/or additional lines for insertion from the same device, the editor has to provide some method of indicating the end of the text file, since it is read first. It is suggested that this should be the pound sign, typed in column 1 on a line by itself. This need not be supplied when there is no risk of confusion, but the editing program always tests for it, so this command would have to be used if any line of text had a pound sign in column 1. (For instance, if it were a file of commands which included insertions with the usual terminator!)

Since the input text file is read immediately after the BEGIN command line, a UI or TI command, if it is to have any effect, must be given on that line. A UC instruction that applies to the first command line after BEGIN must appear on the BEGIN line. A LINE command affecting the reading of the input text must also be given along with BEGIN. If given subsequently, it alters the format for the copying of the file and for the reading and writing of insertions. The user is responsible for the consequences of such a command.

The other option commands may be given either on the BEGIN line or at appropriate points during the editing.

5.b.10 LOOP command (see Examples 4, 5 and 7.B)

5.b.10.a The form and definition of the command are:

LOOP [n] Execute n times the sequence of commands following LOOP. If n is omitted, repetition continues to the end of the file or until a command can no longer be obeyed.

5.b.10.b Implementation of a repeating command

5.b.10.b.1 An implementation of a repeating command that imposes no restrictions on the user and involves no exception to the normal operation of the editor would be ideal. In practice, it has to be accepted that such a command provides more opportunity for error and less chance of quickly detecting it than any other, so that it has to have special conditions attached to it. The aim must be to make these as unobtrusive as possible.

The chief difficulty lies in establishing at an early stage when either an infinite loop has been entered or a finite one is compounding an error. It would be possible, given the DSE, ISAE, ISBE and RSE commands of the present editor, to stipulate that they should be the only string ones acceptable in a LOOP sequence. If this condition were combined with the exclusion of RETURN and RESTART commands, then it could safely be assumed that an error had occurred when the line-pointer had not advanced between two successive loops.

Although temptingly simple, this arrangement would limit the user's freedom too much. It would prevent him, for instance, from replacing the first six out of eight occurrences of "X" in the current line by

```
LOOP 6
RS /X/Y/
*
```

or from making the new file consist of six successive copies of the old by

```
LOOP 5
FL
M
RETURN
*
END
```

Since such operations are reasonable, they should be permitted, so that a higher maximum limit to the number of loops executed must be considered. When operations are being performed on character strings, the limit could be n + 1 (where n is the number of characters in the line). The following pointless looping would then be stopped after a count of 81 (if we assume a line length of 80) rather than

the full 100:

```
LOOP 100
FC /X/
ISA /X/Y/
DS /Y/
*
```

Even if every character of the current line were "X" when the LOOP command is given and an operation performed on each "X", the line-pointer should have moved by the end of the eighty-first loop. The fact that it has not can certainly be interpreted as meaning that something is wrong. The "line-length + 1" limit has, however, no relevance in such cases as the second example above, where the size of the files concerned would have to be known before any logical limit could be set.

5.b.10.c LOOP n (see Example 5)

Because of this last consideration, the editor attempts to execute the specified number of times any LOOP command with a parameter. This may, of course, prove impossible, either for internal reasons or because of the intervention of the operating system - for instance, if a disk file proves too small.

5.b.10.d LOOP (see Examples 4 and 7B)

A LOOP command without a parameter has to be treated differently. By definition, its range of activity is from the current line up to, or towards, the end of the file, so commands involving backward movement - RETURN and RESTART, as well as ERASE and RESET which would be inappropriate in a LOOP sequence - may reasonably be ruled out. This, in turn, allows the use of the "line-length + 1" limit to the number of loops with a stationary line-pointer. With all fear of

infinity eliminated, the remaining problem is to establish with certainty when the LOOP command has been successfully obeyed.

If the end of the file is actually reached, this may be assumed to indicate a satisfactory result. For instance, if the user wants to insert one blank line after every four lines of his 160-line file, he can accomplish this by the sequence

```
LOOP
M3
IA

£
*
```

When a blank line has been inserted after the last line of the file, the command is clearly satisfied.

If, however, the length of the file is 162 lines, the outcome is not so simple. The normal procedure of the editor when a command has failed is to attempt to execute the next command, so that a blank line will be inserted after line 161 and another after line 162 before the end of the file is reached.

Had the LOOP sequence been

```
LOOP
M4
IB

£
*
```

then, no matter the length of the file, the M4 command would eventually fail and the IB command be obeyed "line-length + 1" times.

To avoid such unwanted results, it is necessary to assume success for the LOOP instruction the first time a command in the sequence cannot be obeyed. If an EXIT command is in force, it has its usual effect, otherwise the editor, after giving the appropriate message for the

directive that has failed (in case the outcome is not what the user expects) attempts to execute the command that follows the LOOP sequence.

If the line-pointer is found to have moved backwards during execution of a parameterless LOOP or if it has not advanced after "line-length + 1" loops, an error message (no. 21) is given. The editor stops looping and attempts to execute the directive that follows the LOOP sequence.

5.b.10.e The DSE, ISAE, ISBE and RSE forms of the string commands are related to the LOOP command and have two main purposes.

Firstly, if a change has to take place throughout a line and the relevant string occurs n times, the other ways of accomplishing this are a LOOP n sequence and the issuing of the same string command n times - both rather clumsy methods.

Secondly, the editor provides no sophisticated means of carrying out a change throughout one particular section of the file - except that section from the current line to the end. To limit the area for a repeating change, it is necessary to count the number of times the change is to be performed, so that this number may be used as the parameter of a LOOP n command. The "E" form of the commands means that the counting is only of lines containing particular strings and not of all occurrences of these strings.

Although a change for every occurrence of a character string in a file may be made either with the "E" form of the relevant command or with the form without the "E", the former is more efficient.

Thus

```
LOOP                                    LOOP
FC/X/                                   FC/X/
RSE/X/Y/        is preferable to        RS/X/Y/
*                                       *
```

5.b.10.f The maximum number of lines between the LOOP command and its terminator has been arbitrarily set at ten. If more than ten lines are given, an error message (no. 19) is printed. The editor does not continue to search for the terminator if it has not been found by the eleventh line. Even if it were to appear later, the loop could not be carried out, nor would it be reasonable to allow editing to continue with the command following the LOOP sequence, when so many preceding ones have been ignored. The only available course of action is to assume that an EXIT command is in force, even if one has not been given.

5.b.10.g The nesting of loops has not been included. It is thought that the unnested LOOP and the RESTART directive together give adequate power for most minor repeating amendments and the editor is not intended for operations that can readily be performed by programming languages. For reasons similar to those of the preceding section, an EXIT state is assumed if another LOOP command is found in a LOOP sequence. An error message (no. 20) is given.

5.b.10.h The word "LOOP" was chosen because it is familiar to programmers, already used in the COTAN Amender and shorter and less ambiguous than REPEAT.

The requirement that the first line of a LOOP sequence should not be typed on the same line as the LOOP command is included for consistency with the IA, IB and R instructions.

5.c Information and error messages

The following list is of messages issued by the editor. These are displayed at the console in on-line editing and printed on the line-printer in background jobs. Each message line is preceded by the characters "XXX" to make it readily distinguishable from other material being printed during editing.

Notes are given only when the message is not completely self-explanatory.

1. WHEN "ENTER DATA" APPEARS, START EDITING SESSION BY TYPING

   BEGIN

   This prompt for on-line editing makes no mention of possible option commands, as it is thought this might confuse the inexperienced user.

2. CURRENT LINE IS NO. <line no.>. NO. OF LINES IN FILE IS

   <no. of lines>

   This message is printed, along with the displayed line(s), in response to a P or P+ command.

3. LENGTH OF AMENDED FILE IS <no. of lines> LINES.

4. END OF FILE REACHED. THE ONLY COMMANDS WHICH CAN NOW BE OBEYED

   ARE: END, RESTART, RESET, RETURN, ERASE.

5. END OF FILE REACHED BEFORE THE FOLLOWING COMMAND COULD BE OBEYED

<command>

NO ACTION TAKEN.

After message no. 4 has been given, this will be provoked by any command other than the five named.

6. EXECUTION SUSPENDED BY EXIT COMMAND.

7. FOLLOWING COMMAND NOT RECOGNISED:

<command>

NO ACTION TAKEN.

8. PARAMETER(S) OF FOLLOWING COMMAND MISSING OR NOT RECOGNISED:

<command>

NO ACTION TAKEN.

When an error occurs in the parameter of an option command on a "BEGIN" line, the third line of this message is

EXECUTION TERMINATED.

9. PARAMETER(S) OF FOLLOWING COMMAND GREATER THAN MAXIMUM LENGTH:

<command>

NO ACTION TAKEN.

This applies to character string parameters that exceed the length of the text line. It is not anticipated that this message will be needed very often.

10. PARAMETER(S) OF FOLLOWING COMMAND NOT FOUND IN LINE SPECIFIED:

<command>

NO ACTION TAKEN.

The commands concerned are the various forms of DS, IS and RS.

11. LINE IDENTIFIED BY PARAMETER OF FOLLOWING COMMAND ALREADY CURRENT:

    <command>

    NO ACTION TAKEN.

    This is for use with DB and DC.

12. PARAMETER OF FOLLOWING COMMAND NOT FOUND FROM LINE NO. <line no.>

    TO END OF FILE:

    <command>

    LAST LINE NOW CURRENT. INTERVENING LINES COPIED TO NEW FILE.

    This is used for FB and FC commands.

13. PARAMETER OF FOLLOWING COMMAND NOT FOUND FROM LINE NO. <line no.>

    TO END OF FILE:

    <command>

    LAST LINE NOW CURRENT. INTERVENING LINES DELETED.

    This is used for DB and DC commands.

14. FIRST PARAMETER OF FOLLOWING COMMAND GREATER THAN THE SECOND:

    <command>

    NO ACTION TAKEN.

    The relevant commands are Dm/n and Rm/n.

15. LENGTH OF FILE <no. of lines> LINES. CURRENT LINE NO. <line no.>

    WHEN FOLLOWING COMMAND READ:

    <command>

    NO ACTION TAKEN.

    This message is given after an F, M, D, I or R command with an integer parameter, if the specified movement would go beyond the end of the file. It is used also with DL, if the last line is already current.

16. BEGIN MUST BE THE FIRST EDITING COMMAND.

    EXECUTION TERMINATED.

17. ONLY OPTION COMMANDS RECOGNISED ON "BEGIN" LINE.

    EXECUTION TERMINATED.

18. END OF FILE REACHED BEFORE ALL REPLACEMENT LINES COULD BE ENTERED AFTER R= COMMAND.

    FIRST LINE NOT ENTERED:

    <first unaccepted line>

19. NO. OF LINES AFTER FOLLOWING COMMAND EXCEEDS MAXIMUM:

    LOOP [<n>]

    { EXECUTION SUSPENDED. } (Used in interactive editing)
    { EXECUTION TERMINATED. } (Used in background editing)

20. LOOPS CANNOT BE NESTED.

    { EXECUTION SUSPENDED. } (Used in interactive editing)
    { EXECUTION TERMINATED. } (Used in background editing)

21. FILE POINTER HAS NOT ADVANCED AS EXPECTED IN PARAMETERLESS LOOP.

22. FOLLOWING IS INVALID USE OF THE RESET COMMAND:

    <command>

    NO ACTION TAKEN.

    This would apply if neither of the files had moved as often as specified in the command, since the start of editing or since an operation involving the rewinding of a file.

23. MAXIMUM LENGTH OF FILE LINE IS <maximum no. of characters> CHARACTERS.

    EXECUTION TERMINATED.

    This is used with the LINE command. The present maximum is 132 characters.

CHAPTER 6

PRESENT IMPLEMENTATION AND POSSIBLE EXTENSIONS

Some implementation features have necessarily been discussed in the detailed description of the editor in the preceding chapter. This chapter deals with more technical points about the working of the program - how it operates rather than what it achieves. Some suggestions are made for additional facilities which could usefully be implemented in the future.

6.a Implementation

A general outline of the operation of the editor is given in Figure 1.

6.a.1 Flexibility of program

Although the editing program is primarily for use with the RAX and 44MFT operating systems on an IBM 360 computer, it is intended to be largely machine independent. It is written in FORTRAN IV. Some of the more interesting features of that language - COMMON storage areas, unsubscripted array names in READ/WRITE statements, object-time dimensions for arrays in subroutines, multiply entry and return points in

subroutines, the BACKSPACE statement - could not be incorporated in the program, because of difficulties under RAX. Omission of these elements, however, may make the editor more easily implemented on another machine.

As is partly implied by the existence of the OPTION commands, variables rather than constants have been used for

(1) the dimensions of arrays holding lines of text, character strings, commands

(2) input/output devices

(3) formats used in reading and writing text, insertions, commands and in writing messages

(4) characters used as terminating, continuation and separating symbols.

If ten lines were not a satisfactory length for a LOOP sequence, or a semi-colon not suitable as a command separator, or "80A1" unacceptable as a format for reading a command line, the necessary change could be made by altering specification statements only.

6.a.2 Files and input/output devices

From two to four devices are used by the editor for reading and writing the text file during editing. Normally the text is read from File 1, copied to File 2 (to safeguard the original), then the amended text is written to File 3 during editing and finally copied to File 4 at the end of the editing session. Files 2 and 3 must be tape or disk sequential files, since they may be rewound by the RETURN, RESTART, RESET and ERASE commands. The action of the RESTART command switches these two files, so that File 3 becomes the file being edited and File 2 the amended one. (The interchange is repeated with each subsequent RESTART.) File 4 may be given the

same device number as File 3 and File 1 the same as File 2, but in the latter case the original text will be lost after a RESTART command.

Since the standard procedure of the program is to read and copy the whole of the text file (if File 1 is the same as File 2, no copying is necessary, although the number of lines is counted) before editing starts, it follows that editing commands cannot be interspersed with lines of text in a file. If text and commands are to be read from the same device, the text must be read first, with a separator between that and the command file. (See Section 5.b.9 and the User's guide, Appendix A).

6.a.3 Reading and identifying of commands

It has already been mentioned (Section 5.a) that the comparatively free format for the typing of commands makes their identification rather inefficient. Since blanks are usually significant in character string parameters, it is not even possible to start by removing the blanks in a command line.

In the following account, we assume that EOL is a character immediately after the last character of the line.

The procedure adopted is to examine the first two non-blank characters of the line. (If EOL is the only character found, then the next command line is read.) The first non-blank character by itself does not provide sufficient information for a satisfactory second move.

As far as is practicable, testing is first done for commands which are likely to occur frequently and last for those seldom used. For instance, the simple P command is likely to occur more often than any other. To test at the start for all commands

beginning with P is undesirable, however, since PNEW or PCOMM will probably be rare. Instead, a test is made to see if the second character is EOL or a semi-colon. If it is, the single-letter commands P, M, D, R (all fairly common) are identified in that order at this stage. If the second character is S, then the frequently used commands for altering strings are intercepted. The next tests are for the positioning commands - FB, FC, FL, Fn, Mn - in that order. It is, of course, probable that Fn will occur more often than FL, but the relative inefficiency of testing for a digit makes it more satisfactory to eliminate the letters first.

When the first two non-blank characters have been identified, the next step is clear. With the FB and FC commands, for instance, the next non-blank character is interpreted as a string delimiter and its next occurrence sought. With the FL command, the next non-blank character must be either EOL or a semi-colon. With Fn, the next occurrence of EOL or a semi-colon must be found and the intervening character(s) checked to ensure that an unsigned integer has been specified.

Figure 2 indicates the initial stages in the identification of commands.

The BEGIN command line is treated slightly differently from others. Once BEGIN itself has been identified, tests are made only for LINE, U and T commands. No others are recognised and any type of error will terminate execution.

It may seem unfair to the on-line user not to give him the immediate chance to correct the line. The inexperienced user, however, is unlikely to use any option commands and unlikely, also, to make an error with the single word "BEGIN". The experienced user will have no difficulty in starting a new editing session.

6.a.4 Operations of subroutines

6.a.4.a SHIFT

The subroutine SHIFT is used to find the next non-blank character of a directive or to establish that EOL or a semi-colon, indicating the end of the directive and any parameters, has been reached.

6.a.4.b DIGIT and STRING

When commands requiring parameters have been identified, control is passed to one of two subroutines, DIGIT and STRING. DIGIT checks that the parameters are present and that they are digits, translates them into integers (since the digits were initially read as characters) and returns their integer values. STRING identifies the delimiters of string parameters and stores each parameter in a buffer.

6.a.4.c MOVE

Because the editor is designed mainly for on-line use at a display console and the P + command is expected to be often used, the current line and the following n lines are held in a buffer. The value of n is expressed as a variable in the program and so may be readily altered if the present value of ten is not suitable.

The subroutine MOVE controls this circular buffer, filling it initially and after RETURN, RESTART, ERASE and RESET commands. As the pointer moves to the next line of the text file, the previous current line is replaced in the buffer by the one eleven (or, rather, n + 1) lines farther on in the file.

MOVE is called in any process that causes movement of the file being edited.

It would, of course, be more efficient not to allow the P + command in background editing and so eliminate the need for this buffer. The program is left unaltered so that anybody concerned with the operations of the editor may simulate an on-line session in a background job.

6.a.4.d LINENO

After DIGIT has supplied integer parameters where relevant, the subroutine LINENO executes all commands involving the straightforward counting of lines - Fn, FL, M[n], D[+n], DAn, Dn, Dm/n, DL, IA[n], IB[n], R[+n], Rn, Rm/n. To do this, it calls MOVE.

6.a.4.e LOCATE

The subroutine LOCATE is used to check whether a character string, previously established by STRING, occurs within another string and to set pointers, if it is found.

6.a.4.f FIND

The subroutine FIND, which executes FB, DB, FC and DC commands, makes use of MOVE and, for FC and DC, of LOCATE.

6.a.4.g DELETE and INSERT

Two other subroutines need LOCATE. Subroutine DELETE handles DS, DSE and DSS commands, while INSERT deals with the more complicated execution of IS and RS commands, where expansion and overflow of lines may be involved.

6.a.4.h CURTAB

The simple subroutine CURTAB keeps a table of previous current line numbers, in case a RESET command is given. Each number is reset to zero by a RESET, RETURN, RESTART or ERASE command.

6.a.4.i REPEAT

The subroutine REPEAT is responsible for the execution of a LOOP command. When first called, it stores the LOOP sequence in an array. It maintains a pointer (moved at each subsequent call) to the rows of the array and a count of the times the sequence is obeyed.

6.a.5 Execution of commands not controlled by subroutines

Obeying the LINE command involves not only the straightforward changing of the value of the variable representing the text line length, but the amending of the variable formats for reading the text and for writing it - to a sequential file, to the line-printer or in messages in response to PNEW and PRINT commands.

The execution of the other option commands, control commands and print commands needs little further elaboration on the explanations given in Sections 5.b.6 - 9. Where necessary, additional details are given in the following section.

6.a.6 Flags

6.a.6.a The price paid for the advantages of the PNEW, PRINT, PCOMM, RESET and EXIT commands is the number of tests that must be carried out in the program.

After the execution of a DS, IS or RS command (even if no real change has been made - for instance, with "R$ /X/X/") and during the execution of a complete-line I or R command, a test has to be made for a flag controlled by PNEW and XPNEW.

Each time lines are written to the amended file, a test is made for the PRINT and XPRINT flag.

When a command and its parameters have been established, the PCOMM and XPCOMM flag is checked.

If the line-pointer of either the file being edited or the amended file moves during the execution of a command, the subroutine CURTAB, needed by RESET, must be called.

The EXIT switch needs to be checked only after a command has failed.

6.a.6.b In addition to these flags of convenience, the program needs several more essential ones.

Since the interpretation of the BEGIN command line differs in some respects from the usual one, a test is introduced so that the standard procedure can be followed as far as possible.

When the end of a command line is reached, a LOOP test is needed to establish whether the next line of commands is to be read from the command stream or from the buffer held by the subroutine REPEAT.

If editing has proceeded beyond the last line of the file (for instance by "DL; D" or "FL; IA"), a warning message (no. 4) is given that the only relevant commands are END, RESTART, RESET, RETURN, ERASE. A flag is set "on".

For a command other than these five, a test of this flag is always made as soon as the command has been identified. If the flag is "on", no attempt is made to execute the command. An error message (no. 5) is printed.

6.b Possible extensions to the editor

Although all straightforward editing operations should be practicable with the present program, there are, of course, additional facilities which could save the user time and typing effort or enable him to perform more elaborate operations.

Just which extensions are most needed or desirable depends both on the sophistication of the user and on the type of changes he wants to make. The following list offers one opinion on priorities - in descending order.

6.b.1 It would be useful to be able to limit the area of search more easily and more naturally than by counting the number of times the operation must be performed in order to give a LOOP n command.

Something similar to the MITEM "End" command might be suitable, as suggested in the following examples.

```
E36; FB/XYZ/          Find next line beginning "XYZ" but do not
                      search beyond line no. 36.
EB/GO TO/; FB/GOTO/   Find next line beginning "GOTO" but do not search
                      beyond next line beginning "GO TO".
EC/ABC/; FC/XYZ/      Find next line containing "XYZ" but do not
                      search beyond next line containing "ABC".
EC/ABC/; F119         Find line no. 119 but do not search
                      beyond next line containing "ABC".
E200; LOOP            Replace every occurrence of "X" with "Y"
      FC/X/
      RSE/X/Y         from current line up to line 200.
*
```

In the first four cases above, the effect is clearly a useful "either-or" search.

6.b.2 A simpler way of interchanging blocks of text, rather than by the RETURN command, would be an advantage.

For instance

```
INT 20/26 = 63/81     Interchange the two segments, lines
                      20/26 and lines 63/81.
INT L/ = 63/81        Move lines 63/81 to after the last line.
INT /1 = 63/81        Move lines 63/81 to before first line.
INT B/ABC/C/EFG/ =    Interchange the segment from the line
      63/81           beginning "ABC" to the line containing "EFG"
                      with lines 63/81.
```

6.b.3 A facility for restricting editing to only part of the text line (such as that supplied by the MTS *Ed "Window" command or the "Field" of MITEM) could be helpful in some circumstances.

If the text line were 80 characters in length, the command

LIMIT 21/70

could mean that characters in columns 71 to 80 were not affected by editing - could not overflow because of an insertion command nor move left as the result of a deletion. The search in an FB or DB command (as well as in others, of course) would start in column 21. This might be convenient for a file of tabular data.

6.b.4 The present limited facilities of the U commands could well be extended for the sophisticated user. If a "rewind" instruction were introduced, then the sequence

```
  UO temporary
  IB
First command line
  . . . .
  . . . .
UC normal
*
UO normal
```

would write a sequence of commands on unit no. "temporary", which could then be called as often as required by

UC temporary(rewound)

A block of lines could be inserted at different points of the text file, if it were first written to a separate file and then rewound and inserted at the appropriate places.

A disadvantage of the present insertion of lines is that a terminating character is necessary. If the UA command were to be more generally useful, the IA, IB and R commands could perhaps indicate the

number of lines to be inserted or where the insertion is to end.

For example

IA 12 (11)	After line 12, insert 11 lines from insertion file.
IA 12 (B/ABC/)	After line 12, insert lines from insertion file until line beginning "ABC" is found.
IA 12 (all)	After line 12, insert all lines from insertion file.

6.b.5 At present, if n character string changes have to be repeated throughout the file (or the same part of it) n passes must be made, by means of the RESTART command. If all occurrences of "GOATS" are to be changed to "SHEEP" and of "KIDS" to "LAMBS", it is necessary to give the commands

```
        LOOP
        FC /GOATS/
        RSE /GOATS/SHEEP/
*
        RESTART
        LOOP
        FC /KIDS/
        RSE /KIDS/LAMBS/
*
```

Alternatively, it would be possible to put

```
        LOOP
        RSE /GOATS/SHEEP/
        RSE /KIDS/LAMBS/
        M
*
```

but the price would probably be an outpouring of error messages - unless the words occurred in every line of the text.

A facility for having more than one search active at a time and for having the search areas overlapping would be an improvement. The implementation at Newcastle described by Poole [13] has the advantage that the scope of each change can be simply expressed by line number. Adopting a hybrid notation, we may use the following commands as a self-explanatory illustration

```
1-40        RSE /GOATS/SHEEP/
6           DS /TOO/
8-34        RSE /KIDS/LAMBS/
15          DS /VERY/
30-46       RSE /FENCE/WALL/
```

As the file is read, the relevant active search or searches would be made.

This facility has been very neatly implemented in the University of St Andrews TED Context Editor [2] with its "Exchange" command. This has the form

EX/string1/string2/

As editing continues through the file in accordance with other commands, each line is searched for "string1", which is replaced by "string2". The action of the command can be switched off at the appropriate point in the file by the command

NEX/string1/

Several such searches may be active at one time.

6.b.6 The ability to store character string parameters, or lines of text, or commands in named buffers and to recall them as often as necessary by specifying the buffer names, might in some instances spare the user considerable typing effort.

6.b.7 The same justification may be made for the "tab" facility which some editors provide to save the user from having to count and type spaces in lines for insertion. If the text of the file is in tabular form, such a command could be quite helpful - particularly the "preset" type of command used in MITEM [3], where

T*15 40 65

would define a tab character and set the positions. After that, the line

*MEN*WOMEN*CHILDREN

would be expanded by the program so that the words began in columns 15, 40 and 65 respectively.

The type of command in COTAN[31], on the other hand, would be more useful for the occasional, spaced-out line, since no presetting is required and the column numbers are explicitly given after the tab character, as in

*15MEN*40WOMEN*65CHILDREN

6.b.8 It is sometimes necessary to perform the same operation at a number of different points in the file, yet not in a systematic way that could be incorporated in a LOOP command.

A form of compound command, such as the following, would simplify the process

F(12,B/ABC/,C/XYZ/,L) DS/SHEEP/

This could mean that the first occurrence of "SHEEP" should be deleted on line 12, on the next line beginning "ABC", on the next line containing "XYZ" and on the last line.

6.b.9 It is possible that at the end of an editing session the user does not want to keep the corrected version of his file - that he has been assuring himself that the commands bring about the required changes, but that what he wants to retain are the original and the tested command file. This may be especially likely if the text file is a large one and different versions would need considerable storage.

Preserving the command file is a feature of the QED editor [14] and the Cambridge EDIT [17, 39] and could be considered as a possible addition to this program.

6.b.10 If more advanced facilities were incorporated in the editor, the need for the "Help" command, used by the MTS *Ed [29] and the editor of the University of Manchester MU/MX Operating System [40], might become more pressing! This would display for the on-line user an explanation of one or all of the editing commands.

FLOW DIAGRAMS

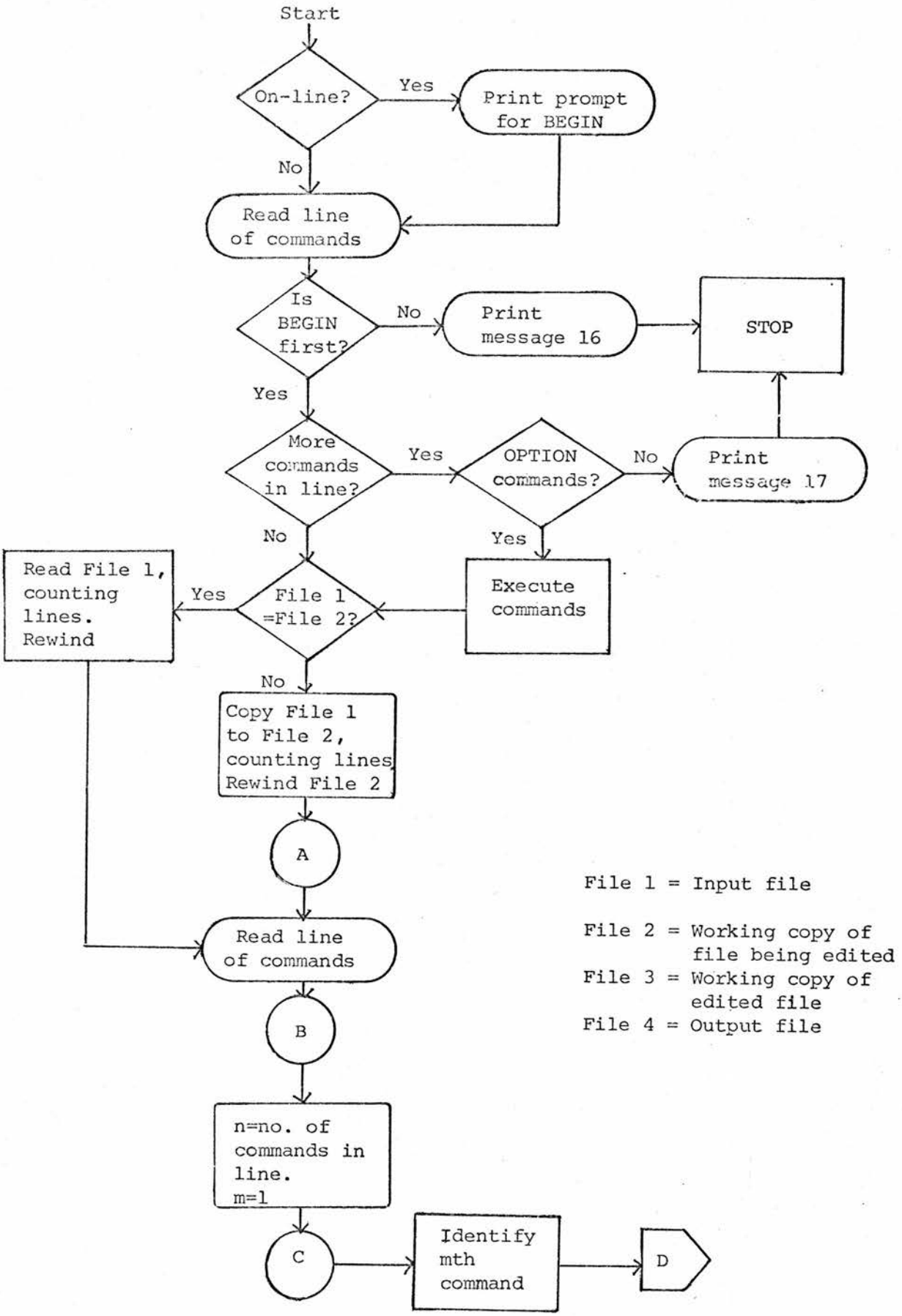

Figure 1 - General Outline of the Editing Program (Page 1)

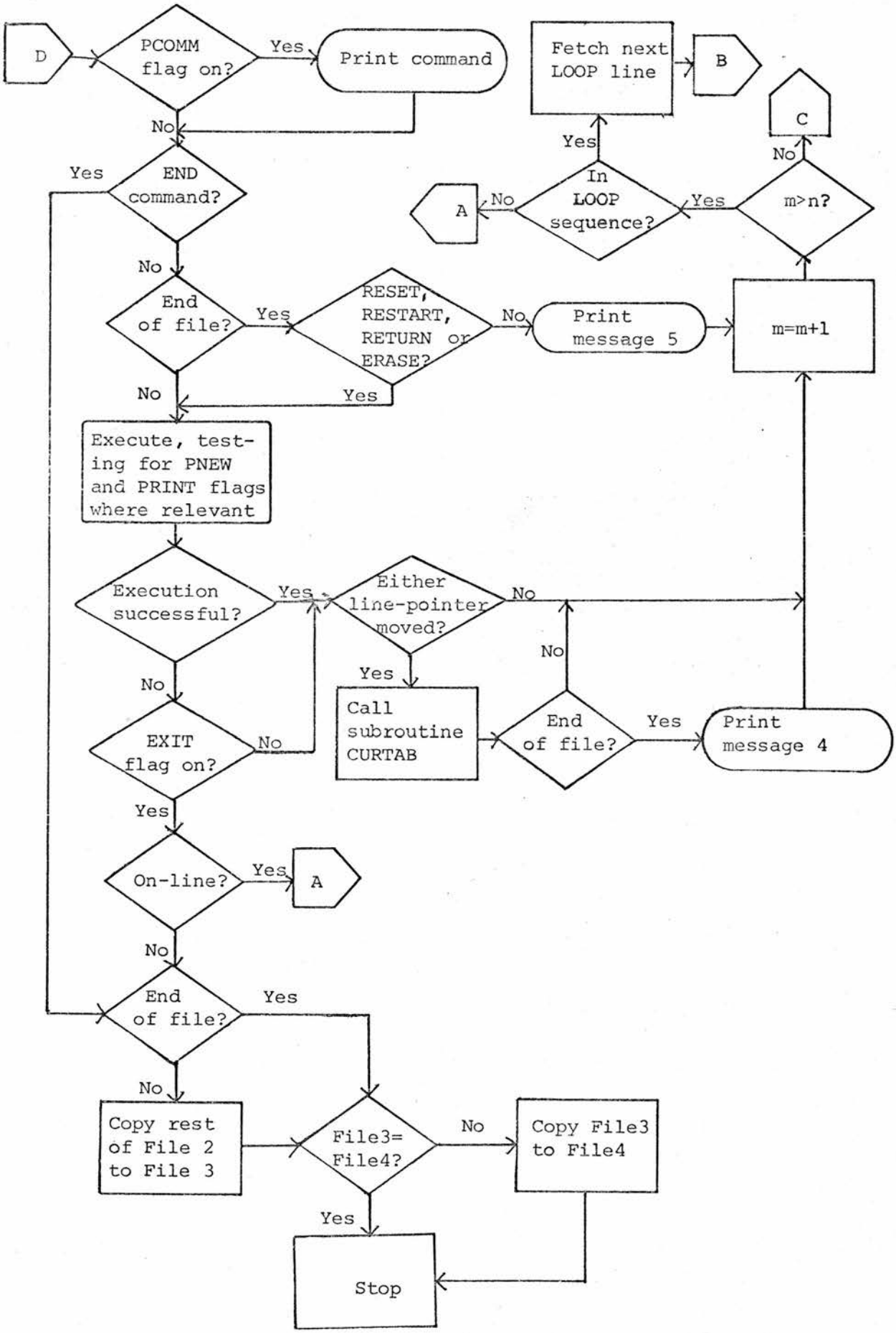

Figure 1 - General Outline of the Editing Program (Page 2)

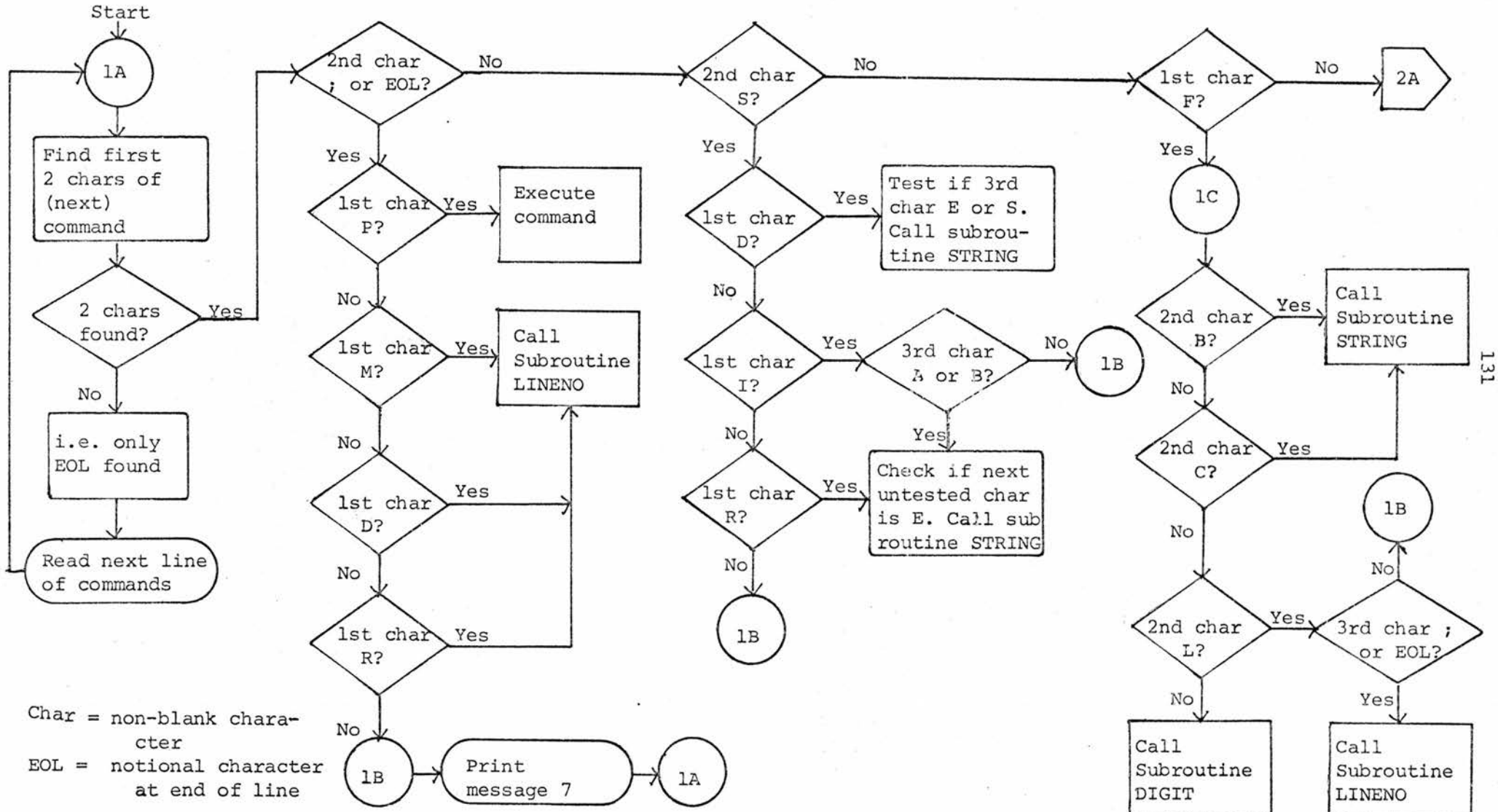

Figure 2 - First Steps in the Identification of Commands (Page 1)

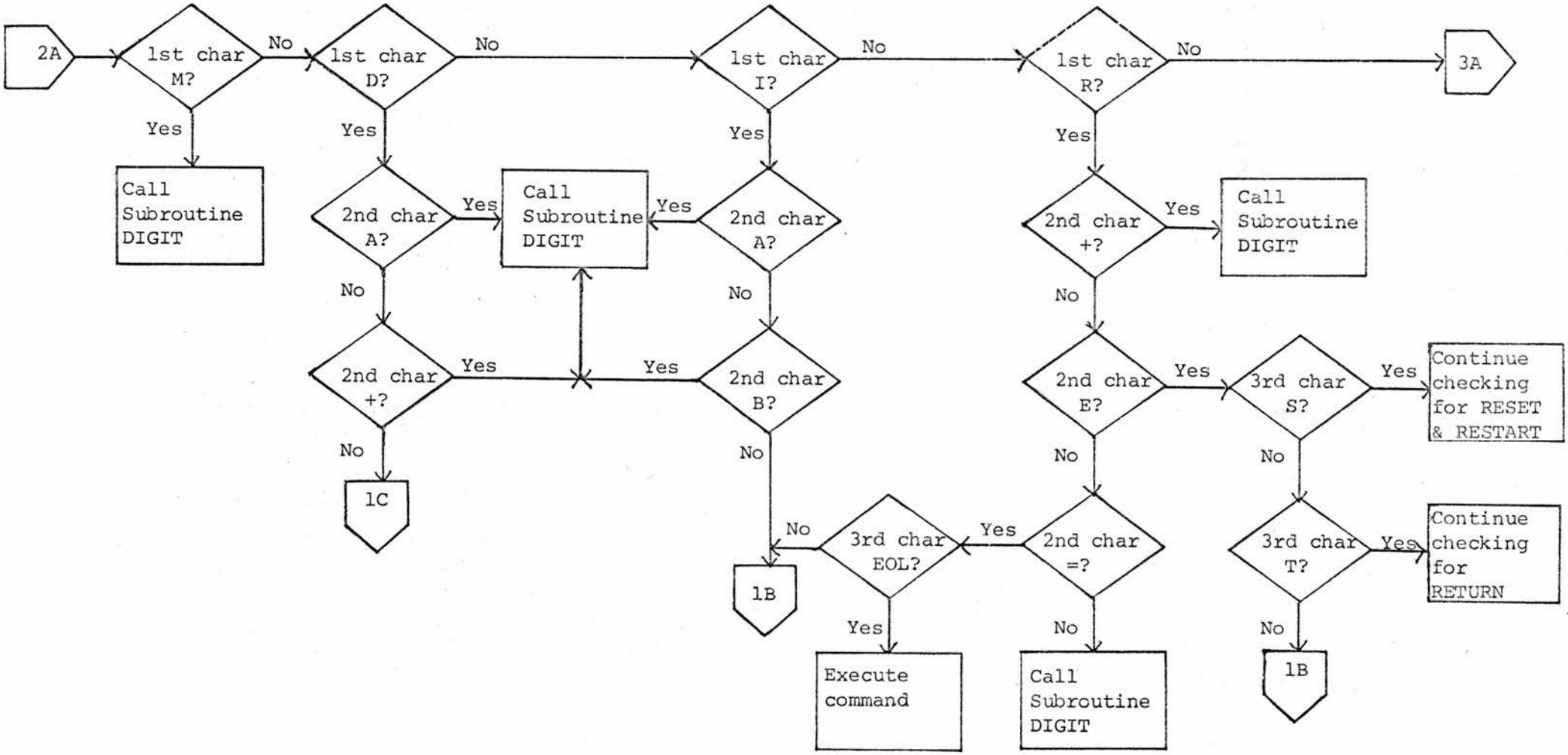

Figure 2 - First Steps in the Identification of Commands (Page 2)

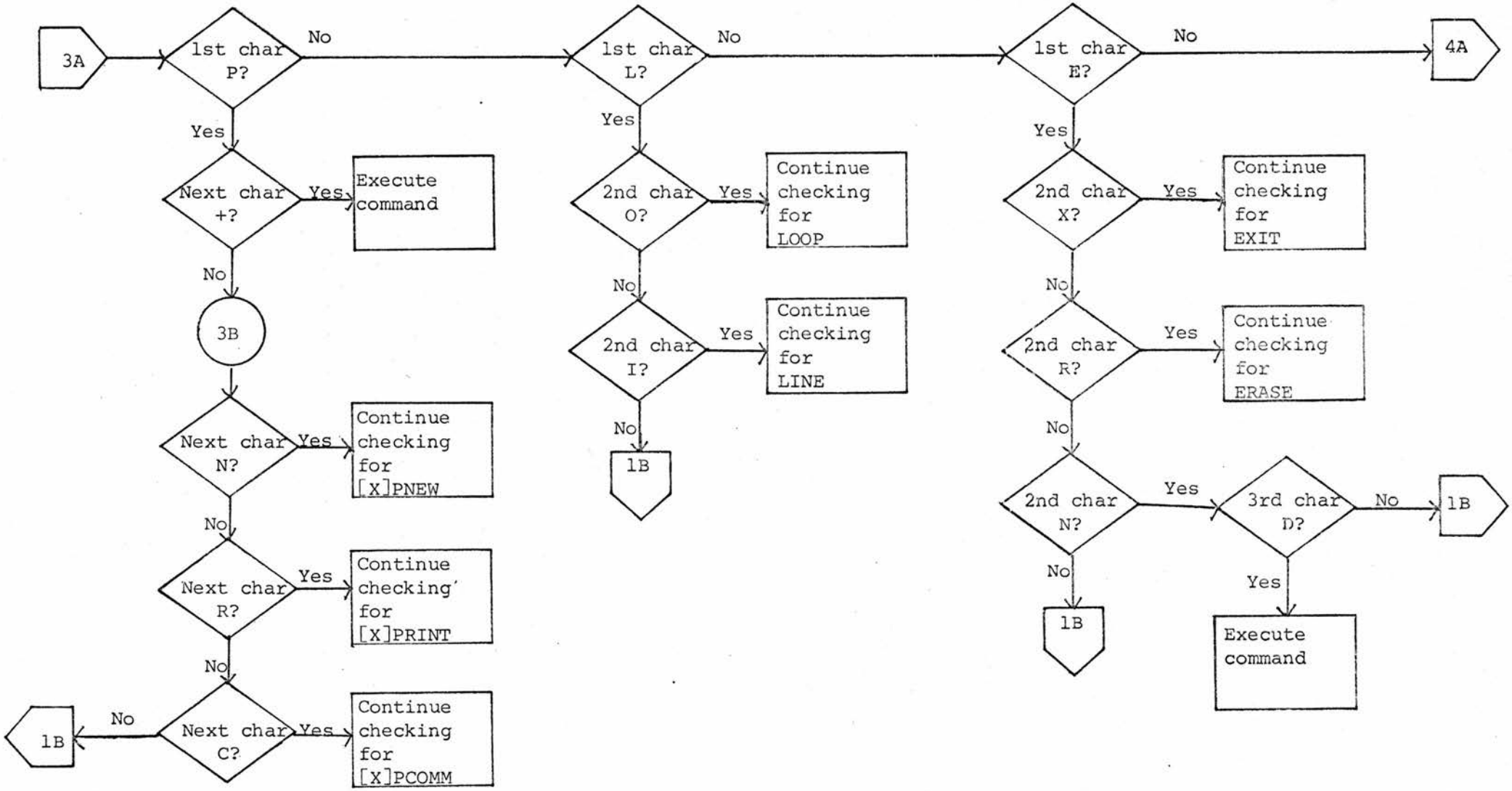

Figure 2 - First Steps in the Identification of Commands (Page 3)

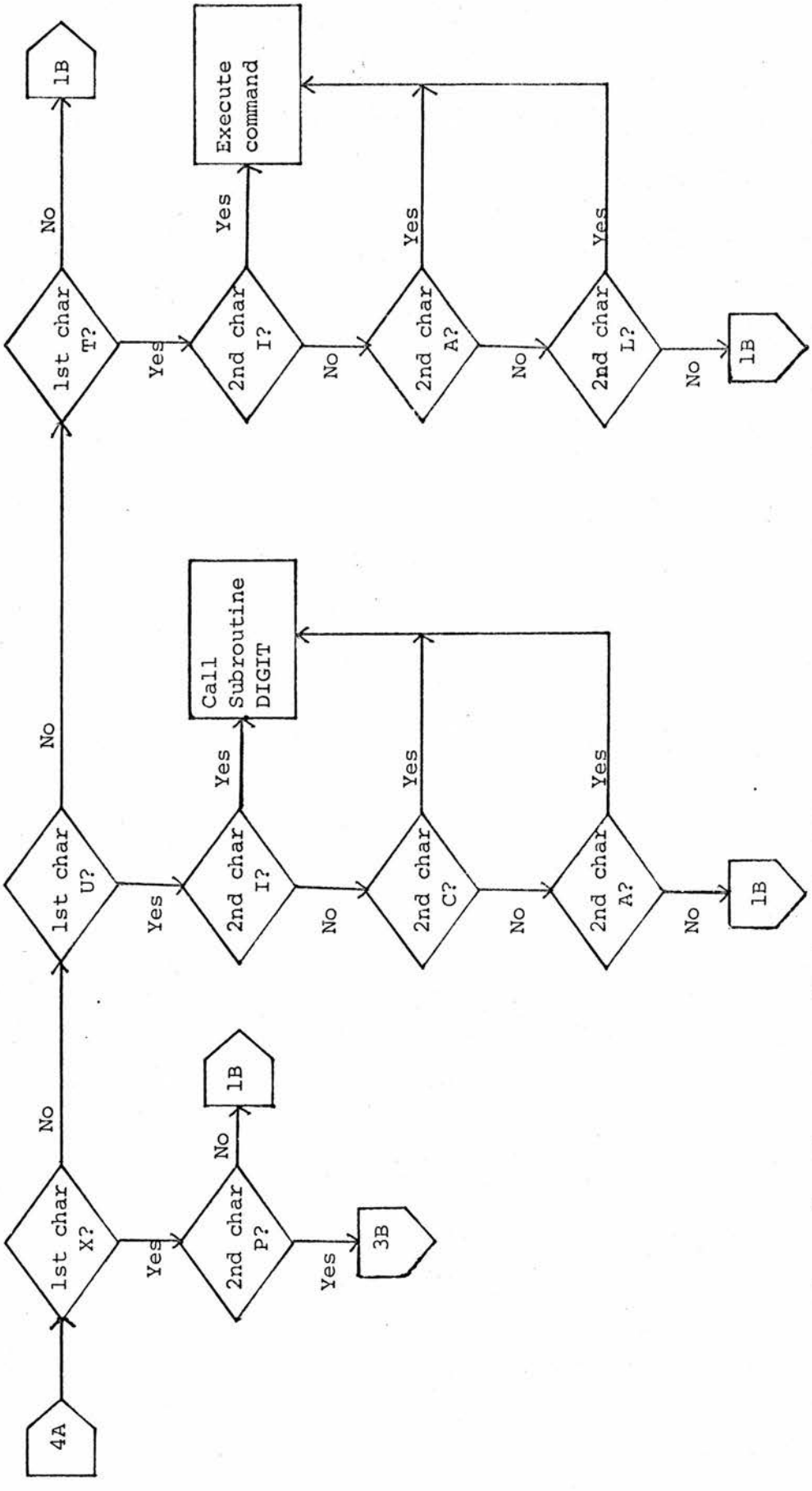

Figure 2 - First Steps in the Identification of Commands (Page 4)

EXAMPLES OF THE EFFECTS OF THE EDITING COMMANDS

The UI, UO, UC and UA commands do not appear in any of the following examples, since their effect cannot be adequately illustrated on paper. Each of the other commands is given in at least one example.

Of the messages printed during the execution of the commands, those preceded by "CCC" are in response to a PCOMM command, those preceded by "NNN" are in response to PNEW, those preceded by "FFF" are in response to PRINT and those preceded by "XXX" are error and information reports.

EXAMPLE 1 'FIND' AND 'MOVE' COMMANDS AND COMMANDS FOR CHANGING THE CURRENT LINE. THE 'LINE' COMMAND IS USED. 'P', 'PCOMM', 'PRINT' AND 'PNEW' ILLUSTRATE THE PROGRESS OF EDITING. NOTE THE EFFECT OF OVERFLOW ON LINES 6, 10 AND 11.

ORIGINAL TEXT

'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
'THERE'S A PORPOISE CLOSE BEHIND US, AND HE'S TREADING ON MY TAIL.
SEE HOW EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
THEY ARE WAITING ON THE SHINGLE -- WILL YOU COME AND JOIN THE DANCE?
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?

'YOU CAN REALLY HAVE NO NOTION HOW DELIGHTFUL IT WILL BE,
WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT TO SEA '
BUT THE SNAIL REPLIED 'TOO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
SAID HE THANKED THE WHITING KINDLY, BUT HE WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, COULD NOT JOIN THE DANCE.

'WHAT MATTERS IT HOW FAR WE GO?' HIS SCALY FRIEND REPLIED.
'THERE IS ANOTHER SHORE, YOU KNOW, UPON THE OTHER SIDE.
THE FURTHER OFF FROM ENGLAND THE NEARER IS TO FRANCE --
THEN TURN NOT PALE, BELOVED SNAIL, BUT COME AND JOIN THE DANCE.
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?

EXAMPLE 1 'FIND' AND 'MOVE' COMMANDS AND COMMANDS FOR CHANGING THE CURRENT LINE

COMMANDS GIVEN

```
BEGIN;  LINE 74
   PCOMM;   PRINT;     PNEW
   P;F1; P;  DS/A /;  F3; P;  ISA/HOW/ EXCITEDLY AND/
 FB/W/;  P;  DSE/ WILL YOU,/; M;P;
   ISAE/WILL/ NOT/
FC/CAN/;  P;    DSS/HO/,/
FB/          W/;  P;  ISB/W/ AT THE MOMENT /
FC/BU/; P;     ISBE /TOO/MUCH, MUCH /
M2;    P;   RS/W/C/;   RSE/CO/WO/
     FL ;  P;   END
```

MESSAGES DURING EXECUTION

```
CCC     PRINT

CCC       PNEW

CCC      P
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    22

CCC  F1

CCC   P
```

EXAMPLE 1 'FIND' AND 'MOVE' COMMANDS AND COMMANDS FOR CHANGING THE CURRENT LINE

```
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    22

CCC     DS/A /
NNN     'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TO A SNAIL.

CCC     F3
FFF     'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TO A SNAIL.
FFF     'THERE'S A PORPOISE CLOSE BEHIND US, AND HE'S TREADING ON MY TAIL.

CCC   P
  SEE HOW EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
XXX  CURRENT LINE IS NO.     3. NO. OF LINES IN FILE IS    22

CCC     ISA/HOW/ EXCITEDLY AND/
NNN     SEE HOW EXCITEDLY AND EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE

CCC     FB/W/
FFF     SEE HOW EXCITEDLY AND EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
FFF     THEY ARE WAITING ON THE SHINGLE -- WILL YOU COME AND JOIN THE DANCE?

CCC     P
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     5. NO. OF LINES IN FILE IS    22

CCC     DSE/ WILL YOU,/
NNN     WON'T YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
```

EXAMPLE 1 'FIND' AND 'MOVE' COMMANDS AND COMMANDS FOR CHANGING THE CURRENT LINE

```
CCC   M
FFF    WON'T YOU, WON'T YOU, WILL YOU JOIN THE DANCE?

CCC  P
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     6. NO. OF LINES IN FILE IS    22

CCC       ISAE/WILL/ NOT/
NNN    WILL NOT YOU, WON'T YOU, WILL NOT YOU, WON'T YOU, WON'T YOU JOIN THE DAN
FFF    WILL NOT YOU, WON'T YOU, WILL NOT YOU, WON'T YOU, WON'T YOU JOIN THE DAN
NNN  CE?

CCC  FC/CAN/
FFF  CE?
FFF
FFF

CCC    P
  'YOU CAN REALLY HAVE NO NOTION HOW DELIGHTFUL IT WILL BE,
XXX  CURRENT LINE IS NO.     9. NO. OF LINES IN FILE IS    22

CCC      DSS/HO/,/
NNN    'YOU CAN REALLY HAVE NO NOTION

CCC  FB/          W/
FFF    'YOU CAN REALLY HAVE NO NOTION

CCC    P
```

EXAMPLE 1 'FIND' AND 'MOVE' COMMANDS AND COMMANDS FOR CHANGING THE CURRENT LINE

```
  WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT TO SEA '
XXX  CURRENT LINE IS NO.    10. NO. OF LINES IN FILE IS    22

CCC    ISB/W/ AT THE MOMENT /
NNN     AT THE MOMENT WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT
FFF     AT THE MOMENT WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT
NNN   TO SEA '

CCC  FC/BU/
FFF   TO SEA '

CCC   P
  BUT THE SNAIL REPLIED 'TCO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
XXX  CURRENT LINE IS NO.    11. NO. OF LINES IN FILE IS    22

CCC      ISBE /TCO/MUCH, MUCH /
NNN    BUT THE SNAIL REPLIED 'MUCH, MUCH TOO FAR, MUCH, MUCH TOO FAR ' AND GAVE
FFF    BUT THE SNAIL REPLIED 'MUCH, MUCH TOO FAR, MUCH, MUCH TOO FAR ' AND GAVE
NNN   A LOOK ASKANCE --

CCC  M2
FFF   A LOOK ASKANCE --
FFF    SAID HE THANKED THE WHITING KINDLY, BUT HE WOULD NOT JOIN THE DANCE.

CCC     P
  WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
XXX  CURRENT LINE IS NO.    13. NO. OF LINES IN FILE IS    22
```

EXAMPLE 1 'FIND' AND 'MOVE' COMMANDS AND COMMANDS FOR CHANGING THE CURRENT LINE

```
CCC     RS/W/C/
NNN    COULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.

CCC     RSE/CO/WO/
NNN    WOULD NOT, WOULD NOT, WOULD NOT, WOULD NOT, WOULD NOT JOIN THE DANCE.

CCC       FL
FFF    WOULD NOT, WOULD NOT, WOULD NOT, WOULD NOT, WOULD NOT JOIN THE DANCE.
FFF    WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, COULD NOT JOIN THE DANCE.
FFF
FFF
FFF    'WHAT MATTERS IT HOW FAR WE GO?' HIS SCALY FRIEND REPLIED.
FFF    'THERE IS ANOTHER SHORE, YOU KNOW, UPON THE OTHER SIDE.
FFF    THE FURTHER OFF FROM ENGLAND THE NEARER IS TO FRANCE --
FFF    THEN TURN NOT PALE, BELOVED SNAIL, BUT COME AND JOIN THE DANCE.
FFF    WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?

CCC    P
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.    22. NO. OF LINES IN FILE IS    22

CCC     END
FFF    WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?

XXX  LENGTH OF AMENDED FILE IS    25 LINES.
```

EXAMPLE 1 'FIND' AND 'MOVE' COMMANDS AND COMMANDS FOR CHANGING THE CURRENT LINE

AMENDED TEXT

'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TO A SNAIL.
'THERE'S A PORPOISE CLOSE BEHIND US, AND HE'S TREADING ON MY TAIL.
SEE HOW EXCITEDLY AND EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
THEY ARE WAITING ON THE SHINGLE -- WILL YOU COME AND JOIN THE DANCE?
WON'T YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL NOT YOU, WON'T YOU, WILL NOT YOU, WON'T YOU, WON'T YOU JOIN THE DAN
CE?

'YOU CAN REALLY HAVE NO NOTION
AT THE MOMENT WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT
TO SEA '
BUT THE SNAIL REPLIED 'MUCH, MUCH TOO FAR, MUCH, MUCH TOO FAR ' AND GAVE
A LOOK ASKANCE --
SAID HE THANKED THE WHITING KINDLY, BUT HE WOULD NOT JOIN THE DANCE.
WOULD NOT, WOULD NOT, WOULD NOT, WOULD NOT, WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, CCULD NOT JOIN THE DANCE.

'WHAT MATTERS IT HOW FAR WE GO?' HIS SCALY FRIEND REPLIED.
'THERE IS ANOTHER SHORE, YOU KNOW, UPON THE OTHER SIDE.
THE FURTHER OFF FROM ENGLAND THE NEARER IS TO FRANCE --
THEN TURN NOT PALE, BELOVED SNAIL, BUT COME AND JOIN THE DANCE.
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?

EXAMPLE 1A AS EXAMPLE 1, WITH 'ERASE', 'XPCOMM', 'XPRINT' AND 'XPNEW' INTRODUCED.
NOTE THAT 'ERASE' DOES NOT AFFECT THE SETTING OF THESE PRINT SWITCHES.

ORIGINAL TEXT AS IN EXAMPLE 1

COMMANDS GIVEN

```
   BEGIN  ;    LINE74
    PCOMM;   PRINT;     PNEW
    P;F1; P;  DS/A /;  F3; P;  ISA/HOW/ EXCITEDLY AND/
    XPRINT
  FB/W/;  P;  DSE/ WILL YOU,/; M;P;
     XPNEW
    ISAE/WILL/ NOT/
     XPCOMM
FC/CAN/;  P;    DSS/HO/,/
FB/          W/;  P;  ISB/W/ AT THE MOMENT /
FC/BU/; P;    ISBE /TOO/MUCH, MUCH /
PCOMM
M2;    P;   RS/W/C/;   RSE/CO/WO/
     FL ;  P;
  ERASE ;P
END
```

EXAMPLE 1A AS EXAMPLE 1, WITH 'ERASE'

MESSAGES DURING EXECUTION

```
CCC     PRINT

CCC       PNEW

CCC      P
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    22

CCC  F1

CCC   P
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    22

CCC    DS/A /
NNN    'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TC A SNAIL.

CCC    F3
FFF    'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TO A SNAIL.
FFF    'THERE'S A PORPOISE CLOSE BEHIND US, AND HE'S TREADING CN MY TAIL.

CCC   P
  SEE HOW EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
XXX  CURRENT LINE IS NO.     3. NO. OF LINES IN FILE IS    22

CCC    ISA/HOW/ EXCITEDLY AND/
```

EXAMPLE 1A AS EXAMPLE 1, WITH 'ERASE'

```
NNN    SEE HOW EXCITEDLY AND EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE

CCC      XPRINT

CCC    FB/W/

CCC    P
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     5. NO. OF LINES IN FILE IS    22

CCC    CSE/ WILL YOU,/
NNN    WON'T YOU, WON'T YOU, WILL YOU JOIN THE DANCE?

CCC   M

CCC  P
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     6. NO. OF LINES IN FILE IS    22

CCC       XPNEW

CCC      ISAE/WILL/ NOT/

CCC       XPCOMM
  'YOU CAN REALLY HAVE NO NOTION HOW DELIGHTFUL IT WILL BE,
XXX  CURRENT LINE IS NO.     9. NO. OF LINES IN FILE IS    22
  WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT TO SEA '
XXX  CURRENT LINE IS NO.    10. NO. OF LINES IN FILE IS    22
```

EXAMPLE 1A AS EXAMPLE 1, WITH 'ERASE'

```
  BUT THE SNAIL REPLIED 'TOO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
XXX  CURRENT LINE IS NO.     11. NO. OF LINES IN FILE IS     22

CCC  M2

CCC       P
  WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
XXX  CURRENT LINE IS NO.     13. NO. OF LINES IN FILE IS     22

CCC      RS/W/C/

CCC      RSE/CO/WO/

CCC        FL

CCC     P
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     22. NO. OF LINES IN FILE IS     22

CCC     ERASE

CCC  P
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.      1. NO. OF LINES IN FILE IS     22

CCC  END

XXX  LENGTH OF AMENDED FILE IS     22 LINES.
```

EXAMPLE 1A AS EXAMPLE 1, WITH 'ERASE'

AMENDED TEXT

'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
'THERE'S A PORPOISE CLOSE BEHIND US, AND HE'S TREADING ON MY TAIL.
SEE HOW EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
THEY ARE WAITING ON THE SHINGLE -- WILL YOU COME AND JOIN THE DANCE?
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?

'YOU CAN REALLY HAVE NO NOTION HOW DELIGHTFUL IT WILL BE,
WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT TO SEA '
BUT THE SNAIL REPLIED 'TOO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
SAID HE THANKED THE WHITING KINDLY, BUT HE WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, COULD NOT JOIN THE DANCE.

'WHAT MATTERS IT HOW FAR WE GO?' HIS SCALY FRIEND REPLIED.
'THERE IS ANOTHER SHORE, YOU KNOW, UPON THE OTHER SIDE.
THE FURTHER OFF FROM ENGLAND THE NEARER IS TO FRANCE --
THEN TURN NOT PALE, BELOVED SNAIL, BUT COME AND JOIN THE DANCE.
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?

EXAMPLE 1B AS EXAMPLE 1A, WITH THE INSERTION OF 'RESTART'.
NOTE THAT 'ERASE' DOES NOT CANCEL THE EFFECT OF 'RESTART'.

ORIGINAL TEXT AS IN EXAMPLE 1

COMMANDS GIVEN

```
  BEGIN; LINE  74
    PCOMM;   PRINT;     PNEW
    P;F1; P;  DS/A /;  F3; P;  ISA/HOW/ EXCITEDLY AND/
  XPRINT
  FB/W/;  P;  DSE/ WILL YOU,/; M;P;
     XPNEW
    ISAE/WILL/ NOT/
 XPCOMM ; RESTART;P
FC/CAN/;  P;    DSS/HO/,/
FB/          W/;  P;  ISB/W/ AT THE MOMENT /
FC/BU/; P;     ISBE /TOO/MUCH, MUCH /
  PCOMM
M2;     P;   RS/W/C/;   RSE/CO/WO/
     FL ;  P;
   ERASE;  P
END
```

EXAMPLE 1B AS EXAMPLE 1A, WITH 'RESTART'

MESSAGES DURING EXECUTION

```
CCC     PRINT

CCC       PNEW

CCC      P
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    22

CCC  F1

CCC   P
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    22

CCC    DS/A /
NNN    'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TO A SNAIL.

CCC    F3
FFF    'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TO A SNAIL.
FFF    'THERE'S A PORPOISE CLOSE BEHIND US, AND HE'S TREADING ON MY TAIL.

CCC   P
  SEE HOW EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
XXX  CURRENT LINE IS NO.     3. NO. OF LINES IN FILE IS    22

CCC    ISA/HOW/ EXCITEDLY AND/
```

EXAMPLE 1B AS EXAMPLE 1A, WITH 'RESTART'

```
NNN    SEE HOW EXCITEDLY AND EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE

CCC    XPRINT

CCC    FB/W/

CCC    P
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     5. NO. OF LINES IN FILE IS    22

CCC    CSE/ WILL YOU,/
NNN    WON'T YOU, WON'T YOU, WILL YOU JOIN THE DANCE?

CCC   M

CCC  P
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     6. NO. OF LINES IN FILE IS    22

CCC       XPNEW

CCC      ISAE/WILL/ NOT/

CCC   XPCOMM
  'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    23
  'YOU CAN REALLY HAVE NO NOTION HOW DELIGHTFUL IT WILL BE,
XXX  CURRENT LINE IS NO.    10. NO. OF LINES IN FILE IS    23
```

EXAMPLE 1B AS EXAMPLE 1A, WITH 'RESTART'

```
  WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT TO SEA '
XXX  CURRENT LINE IS NO.     11. NO. OF LINES IN FILE IS     23
  BUT THE SNAIL REPLIED 'TOO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
XXX  CURRENT LINE IS NO.     12. NO. OF LINES IN FILE IS     23

CCC  M2

CCC       P
  WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
XXX  CURRENT LINE IS NO.     14. NO. OF LINES IN FILE IS     23

CCC      RS/W/C/

CCC      RSE/CO/WO/

CCC        FL

CCC     P
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     23. NO. OF LINES IN FILE IS     23

CCC      ERASE

CCC     P
  'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.      1. NO. OF LINES IN FILE IS     23

CCC  END

XXX  LENGTH OF AMENDED FILE IS     23 LINES.
```

EXAMPLE 1B AS EXAMPLE 1A, WITH 'RESTART'

AMENDED TEXT

'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TO A SNAIL.
'THERE'S A PORPOISE CLOSE BEHIND US, AND HE'S TREADING ON MY TAIL.
SEE HOW EXCITEDLY AND EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
THEY ARE WAITING ON THE SHINGLE -- WILL YOU COME AND JOIN THE DANCE?
WON'T YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL NOT YOU, WON'T YOU, WILL NOT YOU, WON'T YOU, WON'T YOU JOIN THE DAN
CE?

'YOU CAN REALLY HAVE NO NOTION HOW DELIGHTFUL IT WILL BE,
WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT TO SEA '
BUT THE SNAIL REPLIED 'TOO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
SAID HE THANKED THE WHITING KINDLY, BUT HE WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, COULD NOT JOIN THE DANCE.

'WHAT MATTERS IT HOW FAR WE GO?' HIS SCALY FRIEND REPLIED.
'THERE IS ANOTHER SHORE, YOU KNOW, UPON THE OTHER SIDE.
THE FURTHER OFF FROM ENGLAND THE NEARER IS TO FRANCE --
THEN TURN NOT PALE, BELOVED SNAIL, BUT COME AND JOIN THE DANCE.
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?

EXAMPLE 1C AS EXAMPLE 1, WITH 'RESET' INSERTED AND WITHOUT 'PCOMM', 'PRINT' AND 'PNEW'

ORIGINAL TEXT AS IN EXAMPLE 1

COMMANDS GIVEN

```
BEGIN;   LINE 74
    P;F1; P;  DS/A /;  F3; P;  ISA/HOW/ EXCITEDLY AND/
  FB/W/;  P;  DSE/ WILL YOU,/; M;P;
    ISAE/WILL/ NOT/
FC/CAN/;  P;    DSS/HO/,/
FB/          W/;  P;  ISB/W/ AT THE MOMENT /
FC/BU/; P;    ISBE /TOO/MUCH, MUCH /
M2;    P;   RS/W/C/;   RSE/CO/WO/
     FL ;  P;
   RESET; P
END
```

EXAMPLE 1C AS EXAMPLE 1, WITH 'RESET'

MESSAGES DURING EXECUTION

```
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    22
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    22
  SEE HOW EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
XXX  CURRENT LINE IS NO.     3. NO. OF LINES IN FILE IS    22
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     5. NO. OF LINES IN FILE IS    22
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     6. NO. OF LINES IN FILE IS    22
  'YOU CAN REALLY HAVE NO NOTION HOW DELIGHTFUL IT WILL BE,
XXX  CURRENT LINE IS NO.     9. NO. OF LINES IN FILE IS    22
  WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT TO SEA '
XXX  CURRENT LINE IS NO.    10. NO. OF LINES IN FILE IS    22
  BUT THE SNAIL REPLIED 'TOO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
XXX  CURRENT LINE IS NO.    11. NO. OF LINES IN FILE IS    22
  WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
XXX  CURRENT LINE IS NO.    13. NO. OF LINES IN FILE IS    22
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.    22. NO. OF LINES IN FILE IS    22
  WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
XXX  CURRENT LINE IS NO.    13. NO. OF LINES IN FILE IS    22

XXX  LENGTH OF AMENDED FILE IS    25 LINES.
```

EXAMPLE 1C AS EXAMPLE 1, WITH 'RESET'

AMENDED TEXT

'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TO A SNAIL.
'THERE'S A PORPOISE CLOSE BEHIND US, AND HE'S TREADING ON MY TAIL.
SEE HOW EXCITEDLY AND EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
THEY ARE WAITING ON THE SHINGLE -- WILL YOU COME AND JOIN THE DANCE?
WON'T YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL NOT YOU, WON'T YOU, WILL NOT YOU, WON'T YOU, WON'T YOU JOIN THE DAN
CE?

'YOU CAN REALLY HAVE NO NOTION
AT THE MOMENT WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT
TO SEA '
BUT THE SNAIL REPLIED 'MUCH, MUCH TOO FAR, MUCH, MUCH TOO FAR ' AND GAVE
A LOOK ASKANCE --
SAID HE THANKED THE WHITING KINDLY, BUT HE WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, COULD NOT JOIN THE DANCE.

'WHAT MATTERS IT HOW FAR WE GO?' HIS SCALY FRIEND REPLIED.
'THERE IS ANOTHER SHORE, YOU KNOW, UPON THE OTHER SIDE.
THE FURTHER OFF FROM ENGLAND THE NEARER IS TO FRANCE --
THEN TURN NOT PALE, BELOVED SNAIL, BUT COME AND JOIN THE DANCE.
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?

EXAMPLE 1D AS EXAMPLE 1C, WITH 'RESET 1' REPLACING 'RESET'

ORIGINAL TEXT AS IN EXAMPLE 1

COMMANDS GIVEN

```
BEGIN; LINE 74
    P;F1; P;  DS/A /;  F3; P;  ISA/HOW/ EXCITEDLY AND/
  FB/W/;  P;  DSE/ WILL YOU,/; M;P;
    ISAE/WILL/ NOT/
FC/CAN/;  P;    DSS/HO/,/
FB/          W/;  P;  ISB/W/ AT THE MOMENT /
FC/BU/; P;    ISBE /TOO/MUCH, MUCH /
M2;    P;   RS/W/C/;   RSE/CO/WO/
     FL ;  P;
RESET 1;P
END
```

MESSAGES DURING EXECUTION AS IN EXAMPLE 1C

AMENDED TEXT AS IN EXAMPLE 1C

EXAMPLE 1E AS EXAMPLE 1C, WITH 'RESET 2' REPLACING 'RESET'

ORIGINAL TEXT AS IN EXAMPLE 1

COMMANDS GIVEN

```
BEGIN; LINE 74
    P;F1; P;  DS/A /;  F3; P;  ISA/HOW/ EXCITEDLY AND/
  FB/W/;  P;  DSE/ WILL YOU,/; M;P;
    ISAE/WILL/ NOT/
FC/CAN/;  P;    DSS/HO/,/
FB/          W/;  P;  ISB/W/ AT THE MOMENT /
FC/BU/; P;    ISBE /TOO/MUCH, MUCH /
M2;    P;   RS/W/C/;   RSE/CO/WO/
     FL ;  P;
   RESET 2;  P
END
```

EXAMPLE 1E AS EXAMPLE 1C, WITH 'RESET 2' REPLACING 'RESET'

MESSAGES DURING EXECUTION

```
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    22
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    22
  SEE HOW EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
XXX  CURRENT LINE IS NO.     3. NO. OF LINES IN FILE IS    22
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     5. NO. OF LINES IN FILE IS    22
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     6. NO. OF LINES IN FILE IS    22
  'YOU CAN REALLY HAVE NO NOTION HOW DELIGHTFUL IT WILL BE,
XXX  CURRENT LINE IS NO.     9. NO. OF LINES IN FILE IS    22
  WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT TO SEA '
XXX  CURRENT LINE IS NO.    10. NO. OF LINES IN FILE IS    22
  BUT THE SNAIL REPLIED 'TOO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
XXX  CURRENT LINE IS NO.    11. NO. OF LINES IN FILE IS    22
  WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
XXX  CURRENT LINE IS NO.    13. NO. OF LINES IN FILE IS    22
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.    22. NO. OF LINES IN FILE IS    22
  BUT THE SNAIL REPLIED 'TOO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
XXX  CURRENT LINE IS NO.    11. NO. OF LINES IN FILE IS    22

XXX  LENGTH OF AMENDED FILE IS    25 LINES.
```

EXAMPLE 1E AS EXAMPLE 1C, WITH 'RESET 2' REPLACING 'RESET'

AMENDED TEXT

'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TO A SNAIL.
'THERE'S A PORPOISE CLOSE BEHIND US, AND HE'S TREADING ON MY TAIL.
SEE HOW EXCITEDLY AND EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
THEY ARE WAITING ON THE SHINGLE -- WILL YOU COME AND JOIN THE DANCE?
WON'T YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL NOT YOU, WON'T YOU, WILL NOT YOU, WON'T YOU, WON'T YOU JOIN THE DAN
CE?

'YOU CAN REALLY HAVE NO NOTION
AT THE MOMENT WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT
TO SEA '
BUT THE SNAIL REPLIED 'MUCH, MUCH TOO FAR, MUCH, MUCH TOO FAR ' AND GAVE
BUT THE SNAIL REPLIED 'TOO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
SAID HE THANKED THE WHITING KINDLY, BUT HE WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, COULD NOT JOIN THE DANCE.

'WHAT MATTERS IT HOW FAR WE GO?' HIS SCALY FRIEND REPLIED.
'THERE IS ANOTHER SHORE, YOU KNOW, UPON THE OTHER SIDE.
THE FURTHER OFF FROM ENGLAND THE NEARER IS TO FRANCE --
THEN TURN NOT PALE, BELOVED SNAIL, BUT COME AND JOIN THE DANCE.
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?

EXAMPLE 1F AS EXAMPLE 1C, WITH 'RESET 3' REPLACING 'RESET'

ORIGINAL TEXT AS IN EXAMPLE 1

COMMANDS GIVEN

```
BEGIN; LINE 74
    P;F1; P;  DS/A /;  F3; P;  ISA/HOW/ EXCITEDLY AND/
  FB/W/;  P;  DSE/ WILL YOU,/; M;P;
    ISAE/WILL/ NOT/
FC/CAN/;  P;    DSS/HO/,/
FB/         W/;  P;  ISB/W/ AT THE MOMENT /
FC/BU/; P;    ISBE /TOO/MUCH, MUCH /
M2;    P;   RS/W/C/;   RSE/CO/WO/
     FL ;  P;
RESET3;P
END
```

EXAMPLE 1F AS EXAMPLE 1C, WITH 'RESET 3' REPLACING 'RESET'

MESSAGES DURING EXECUTION

```
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    22
  'WILL YOU WALK A LITTLE FASTER?' SAID A WHITING TO A SNAIL.
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    22
  SEE HOW EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
XXX  CURRENT LINE IS NO.     3. NO. OF LINES IN FILE IS    22
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     5. NO. OF LINES IN FILE IS    22
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.     6. NO. OF LINES IN FILE IS    22
  'YOU CAN REALLY HAVE NO NOTION HOW DELIGHTFUL IT WILL BE,
XXX  CURRENT LINE IS NO.     9. NO. OF LINES IN FILE IS    22
  WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT TO SEA '
XXX  CURRENT LINE IS NO.    10. NO. OF LINES IN FILE IS    22
  BUT THE SNAIL REPLIED 'TOO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
XXX  CURRENT LINE IS NO.    11. NO. OF LINES IN FILE IS    22
  WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
XXX  CURRENT LINE IS NO.    13. NO. OF LINES IN FILE IS    22
  WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?
XXX  CURRENT LINE IS NO.    22. NO. OF LINES IN FILE IS    22
  BUT THE SNAIL REPLIED 'TOO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
XXX  CURRENT LINE IS NO.    11. NO. OF LINES IN FILE IS    22

XXX  LENGTH OF AMENDED FILE IS    24 LINES.
```

EXAMPLE 1F AS EXAMPLE 1C, WITH 'RESET 3' REPLACING 'RESET'

AMENDED TEXT

'WILL YOU WALK LITTLE FASTER?' SAID A WHITING TO A SNAIL.
'THERE'S A PORPOISE CLOSE BEHIND US, AND HE'S TREADING ON MY TAIL.
SEE HOW EXCITEDLY AND EAGERLY THE LOBSTERS AND THE TURTLES ALL ADVANCE
THEY ARE WAITING ON THE SHINGLE -- WILL YOU COME AND JOIN THE DANCE?
WON'T YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL NOT YOU, WON'T YOU, WILL NOT YOU, WON'T YOU, WON'T YOU JOIN THE DAN-
CE?

'YOU CAN REALLY HAVE NO NOTION
AT THE MOMENT WHEN THEY TAKE US UP AND THROW US, WITH THE LOBSTERS, OUT
TO SEA '
BUT THE SNAIL REPLIED 'TOO FAR, TOO FAR ' AND GAVE A LOOK ASKANCE --
SAID HE THANKED THE WHITING KINDLY, BUT HE WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, WOULD NOT JOIN THE DANCE.
WOULD NOT, COULD NOT, WOULD NOT, COULD NOT, COULD NOT JOIN THE DANCE.

'WHAT MATTERS IT HOW FAR WE GO?' HIS SCALY FRIEND REPLIED.
'THERE IS ANOTHER SHORE, YOU KNOW, UPON THE OTHER SIDE.
THE FURTHER OFF FROM ENGLAND THE NEARER IS TO FRANCE --
THEN TURN NOT PALE, BELOVED SNAIL, BUT COME AND JOIN THE DANCE.
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WILL YOU JOIN THE DANCE?
WILL YOU, WON'T YOU, WILL YOU, WON'T YOU, WON'T YOU JOIN THE DANCE?

EXAMPLE 2 DELETION OF COMPLETE LINES. 'P' AND 'PCOMM' ILLUSTRATE THE PROGRESS OF EDITING.

ORIGINAL TEXT

```
THEY TOLD ME YOU HAD BEEN TO HER,                1
  AND MENTIONED ME TO HIM,                       2
SHE GAVE ME A GOOD CHARACTER,                    3
  BUT SAID I COULD NOT SWIM.                     4
                                                 5
HE SENT THEM WORD I HAD NOT GONE,                6
  (WE KNEW IT TO BE TRUE);                       7
IF SHE SHOULD PUSH THE MATTER ON,                8
  WHAT WOULD BECOME OF YOU?                      9
                                                10
I GAVE HER ONE, THEY GAVE HIM TWO,               11
  YOU GAVE US THREE OR MORE;                     12
THEY ALL RETURNED FROM HIM TO YOU,               13
  THOUGH THEY WERE MINE BEFORE.                  14
                                                15
IF I OR SHE SHOULD CHANCE TO BE                  16
  INVOLVED IN THIS AFFAIR,                       17
HE TRUSTS TO YOU TO SET THEM FREE,               18
  EXACTLY AS WE WERE.                            19
                                                20
MY NOTION WAS THAT YOU HAD BEEN                  21
  (BEFORE SHE HAD THIS FIT)                      22
AN OBSTACLE THAT CAME BETWEEN                    23
  HIM, AND OURSELVES, AND IT.                    24
```

EXAMPLE 2 DELETION OF COMPLETE LINES

COMMANDS GIVEN

```
BEGIN
   PCOMM
     D2; P;  D4/5;   P;    D+2;    P
DB/I/;   P;    M;P; DA16;    P;    DC/AS /;   P
D    ;   P;    DL ;    P;    END
```

MESSAGES DURING EXECUTION

```
CCC       D2

CCC   P
    SHE GAVE ME A GOOD CHARACTER,                    3
XXX  CURRENT LINE IS NO.     3. NO. OF LINES IN FILE IS    24

CCC    D4/5

CCC     P
    HE SENT THEM WORD I HAD NOT GONE,                6
XXX  CURRENT LINE IS NO.     6. NO. OF LINES IN FILE IS    24

CCC      D+2

CCC      P
```

EXAMPLE 2 DELETION OF COMPLETE LINES

```
      WHAT WOULD BECOME OF YCU?                                  9
XXX   CURRENT LINE IS NO.     9. NO. OF LINES IN FILE IS     24

CCC   DB/I/

CCC      P
     I GAVE HER ONE, THEY GAVE HIM TWO,                         11
XXX   CURRENT LINE IS NO.    11. NO. OF LINES IN FILE IS     24

CCC       M

CCC   P
      YOU GAVE US THREE OR MORE;                                 12
XXX   CURRENT LINE IS NO.    12. NO. OF LINES IN FILE IS     24

CCC    DA16

CCC       P
      INVOLVED IN THIS AFFAIR,                                   17
XXX   CURRENT LINE IS NO.    17. NO. OF LINES IN FILE IS     24

CCC       DC/AS /

CCC      P
      EXACTLY AS WE WERE.                                        19
XXX   CURRENT LINE IS NO.    19. NO. OF LINES IN FILE IS     24
```

EXAMPLE 2 DELETION OF COMPLETE LINES

```
CCC  D

CCC     P

                                                          20
XXX  CURRENT LINE IS NO.     20. NO. OF LINES IN FILE IS     24

CCC      DL

CCC      P
       HIM, AND OURSELVES, AND IT.                        24
XXX  CURRENT LINE IS NO.     24. NO. OF LINES IN FILE IS     24

CCC      END

XXX  LENGTH OF AMENDED FILE IS      4 LINES.
```

AMENDED TEXT

```
    THEY TOLD ME YOU HAD BEEN TO HER,                    1
    SHE GAVE ME A GOOD CHARACTER,                        3
    I GAVE HER ONE, THEY GAVE HIM TWO,                  11
      HIM, AND OURSELVES, AND IT.                       24
```

EXAMPLE 3 INSERTION AND REPLACEMENT OF COMPLETE LINES.
THE 'TA' COMMAND IS SHOWN. 'P', 'P+', 'PCOMM' AND 'PNEW'
ILLUSTRATE THE PROGRESS OF EDITING.

ORIGINAL TEXT

```
OH, GENTLEMEN, LISTEN, I PRAY,                    1
THOUGH I OWN THAT MY HEART HAS BEEN RANGING       2
OF NATURE THE LAWS I OBEY,                        3
FOR NATURE IS CONSTANTLY CHANGING.                4
THE MOON IN HER PHASES IS FOUND,                  5
THE TIME AND THE WIND AND THE WEATHER,            6
THE MONTHS IN SUCCESSION COME ROUND,              7
AND YOU DON'T FIND TWO MONDAYS TOGETHER.          8
 CONSIDER THE MORAL, I PRAY,                      9
 NOR BRING A YOUNG FELLOW TO SORROW,             10
 WHO LOVES THIS YOUNG LADY TODAY,                11
 AND LOVES THAT YOUNG LADY TOMORROW.             12
                                                 13
                                                 14
YOU CANNOT EAT BREAKFAST ALL DAY,                15
NOR IS IT THE ACT OF A SINNER,                   16
WHEN BREAKFAST IS TAKEN AWAY,                    17
TO TURN YOUR ATTENTION TO DINNER;                18
AND IT'S NOT IN THE RANGE OF BELIEF,             19
THAT YOU COULD HOLD HIM AS A GLUTTON,            20
WHO, WHEN HE IS TIRED OF BEEF,                   21
DETERMINES TO TACKLE THE MUTTON.                 22
 BUT THIS I AM WILLING TO SAY,                   23
 IF IT WILL APPEASE HER SORROW,                  24
 I'LL MARRY THIS LADY TODAY,                     25
 AND I'LL MARRY THE OTHER TOMORROW.              26
```

EXAMPLE 3 INSERTION AND REPLACEMENT OF COMPLETE LINES AND 'TA'

COMMANDS GIVEN

```
BEGIN
    PCOMM;PNEW
P;IB
      INSERTION BEFORE LINE 1
£
   P;   IA;
      INSERTION AFTER LINE 1
£
   P;   IB4
      FIRST INSERTION BEFORE LINE 4
      SECCND INSERTION BEFORE LINE 4
£
    P;   IA5
      FIRST INSERTION AFTER LINE 5
      SECOND INSERTION AFTER LINE 5
      THIRD INSERTION AFTER LINE 5
£
    P;  R
      FIRST LINE TO REPLACE LINE 6
      SECOND LINE TO REPLACE LINE 6
£
   P;   R10
      FIRST LINE TO REPLACE LINE 10
      SECOND LINE TO REPLACE LINE 10
      THIRD LINE TO REPLACE LINE 10
      FOURTH LINE TO REPLACE LINE 10
£
```

EXAMPLE 3 INSERTION AND REPLACEMENT OF COMPLETE LINES AND 'TA'

```
    P; R12/13
      REPLACEMENT FOR LINES 12-13
£
    P; M; P;  R=
      THIS SHOULD REPLACE LINE 15
      THIS SHOULD REPLACE LINE 16
      THIS SHOULD REPLACE LINE 17
      THIS SHOULD REPLACE LINE 18
£
P+;  M;  P;  TA+;  R+1
      REPLACEMENT FOR LINES 20-21
+
  END
```

MESSAGES DURING EXECUTION

```
 CCC  PNEW

 CCC  P
     OH, GENTLEMEN, LISTEN, I PRAY,                    1
 XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS    26

 CCC  IB
 NNN         INSERTION BEFORE LINE 1
```

EXAMPLE 3 INSERTION AND REPLACEMENT OF COMPLETE LINES AND 'TA'

```
CCC     P
    OH, GENTLEMEN, LISTEN, I PRAY,                          1
XXX  CURRENT LINE IS NO.     1. NO. OF LINES IN FILE IS     26

CCC     IA
NNN        INSERTION AFTER LINE 1

CCC     P
    THOUGH I OWN THAT MY HEART HAS BEEN RANGING             2
XXX  CURRENT LINE IS NO.     2. NO. OF LINES IN FILE IS     26

CCC     IB4
NNN        FIRST INSERTION BEFORE LINE 4
NNN        SECOND INSERTION BEFORE LINE 4

CCC      P
    FOR NATURE IS CONSTANTLY CHANGING.                      4
XXX  CURRENT LINE IS NO.     4. NO. OF LINES IN FILE IS     26

CCC     IA5
NNN        FIRST INSERTION AFTER LINE 5
NNN        SECOND INSERTION AFTER LINE 5
NNN        THIRD INSERTION AFTER LINE 5

CCC      P
    THE TIME AND THE WIND AND THE WEATHER,                  6
XXX  CURRENT LINE IS NO.     6. NO. OF LINES IN FILE IS     26
```

EXAMPLE 3 INSERTION AND REPLACEMENT OF COMPLETE LINES AND 'TA'

```
CCC    R
NNN        FIRST LINE TO REPLACE LINE 6
NNN        SECOND LINE TO REPLACE LINE 6

CCC     P
    THE MONTHS IN SUCCESSION COME ROUND,                7
XXX  CURRENT LINE IS NO.     7. NO. OF LINES IN FILE IS    26

CCC     R10
NNN        FIRST LINE TO REPLACE LINE 10
NNN        SECOND LINE TO REPLACE LINE 10
NNN        THIRD LINE TO REPLACE LINE 10
NNN        FOURTH LINE TO REPLACE LINE 10

CCC      P
     WHO LOVES THIS YOUNG LADY TODAY,                  11
XXX  CURRENT LINE IS NO.    11. NO. OF LINES IN FILE IS    26

CCC   R12/13
NNN        REPLACEMENT FOR LINES 12-13

CCC      P
                                                       14
XXX  CURRENT LINE IS NO.    14. NO. OF LINES IN FILE IS    26

CCC   M
```

EXAMPLE 3 INSERTION AND REPLACEMENT OF COMPLETE LINES AND 'TA'

```
CCC   P
    YOU CANNOT EAT BREAKFAST ALL DAY,                  15
XXX  CURRENT LINE IS NO.    15. NO. OF LINES IN FILE IS    26

CCC    R=
NNN         THIS SHOULD REPLACE LINE 15
NNN         THIS SHOULD REPLACE LINE 16
NNN         THIS SHOULD REPLACE LINE 17
NNN         THIS SHOULD REPLACE LINE 18

CCC  P+
    AND IT'S NOT IN THE RANGE OF BELIEF,               19
    THAT YOU COULD HOLD HIM AS A GLUTTON,              20
    WHO, WHEN HE IS TIRED OF BEEF,                     21
    DETERMINES TO TACKLE THE MUTTON.                   22
     BUT THIS I AM WILLING TO SAY,                     23
     IF IT WILL APPEASE HER SORROW,                    24
     I'LL MARRY THIS LADY TODAY,                       25
     AND I'LL MARRY THE OTHER TOMORROW.                26


XXX  CURRENT LINE IS NO.    19. NO. OF LINES IN FILE IS    26

CCC    M
```

EXAMPLE 3 INSERTION AND REPLACEMENT OF COMPLETE LINES AND 'TA'

```
CCC    P
    THAT YOU COULD HOLD HIM AS A GLUTTON,              20
XXX  CURRENT LINE IS NO.    20. NO. OF LINES IN FILE IS    26

CCC    TA+

CCC    R+1
NNN        REPLACEMENT FOR LINES 20-21

CCC    END

XXX  LENGTH OF AMENDED FILE IS    35 LINES.
```

AMENDED TEXT

```
  INSERTION BEFORE LINE 1
OH, GENTLEMEN, LISTEN, I PRAY,                          1
  INSERTION AFTER LINE 1
THOUGH I OWN THAT MY HEART HAS BEEN RANGING             2
OF NATURE THE LAWS I OBEY,                              3
  FIRST INSERTION BEFORE LINE 4
  SECOND INSERTION BEFORE LINE 4
FOR NATURE IS CONSTANTLY CHANGING.                      4
THE MOON IN HER PHASES IS FOUND,                        5
  FIRST INSERTION AFTER LINE 5
  SECOND INSERTION AFTER LINE 5
  THIRD INSERTION AFTER LINE 5
```

EXAMPLE 3 INSERTION AND REPLACEMENT OF COMPLETE LINES AND 'TA'

```
  FIRST LINE TO REPLACE LINE 6
  SECOND LINE TO REPLACE LINE 6
THE MONTHS IN SUCCESSION COME ROUND,                7
AND YOU DON'T FIND TWO MONDAYS TOGETHER.            8
 CONSIDER THE MORAL, I PRAY,                        9
  FIRST LINE TO REPLACE LINE 10
  SECOND LINE TO REPLACE LINE 10
  THIRD LINE TO REPLACE LINE 10
  FOURTH LINE TO REPLACE LINE 10
 WHO LOVES THIS YOUNG LADY TODAY,                   11
  REPLACEMENT FOR LINES 12-13
                                                    14
  THIS SHOULD REPLACE LINE 15
  THIS SHOULD REPLACE LINE 16
  THIS SHOULD REPLACE LINE 17
  THIS SHOULD REPLACE LINE 18
AND IT'S NOT IN THE RANGE OF BELIEF,                19
  REPLACEMENT FOR LINES 20-21
DETERMINES TO TACKLE THE MUTTON.                    22
 BUT THIS I AM WILLING TO SAY,                      23
 IF IT WILL APPEASE HER SORROW,                     24
 I'LL MARRY THIS LADY TODAY,                        25
 AND I'LL MARRY THE OTHER TOMORROW.                 26
```

EXAMPLE 4 PARAMETERLESS 'LOOP' AND 'RESTART' ARE USED TO CHANGE 'J' TO 'JJ' AND 'MARK' TO 'NN' THROUGHOUT FILE. THE 'TL' COMMAND IS SHOWN. 'PCOMM' AND 'PNEW' ILLUSTRATE THE PROGRESS OF EDITING. (SEE ALSO EXAMPLE 7B)

ORIGINAL TEXT

```
      DIMENSION NAME(100,16),MARK(100),MTEMP(16)
      READ(5,10)N
10    FORMAT(I6)
      READ(5,30)((NAME(I,J),J=1,16),MARK(I),I=1,N)
30    FORMAT(16A4,I6)
      WRITE(6,40)((NAME(I,J),J=1,16),MARK(I),I=1,N)
40    FORMAT(' ',16A4,I6)
      L=N-1
50    I=1
60    IF(MARK(I).GE.MARK(I+1))GO TO 80
      M=MARK(I)
      MARK(I)=MARK(I+1)
      MARK(I+1)=M
      DO 70 J=1,16
      MTEMP(J)=NAME(I,J)
      NAME(I,J)=NAME(I+1,J)
70    NAME(I+1,J)=MTEMP(J)
80    I=I+1
      IF(I.LE.L)GO TO 60
      L=L-1
      IF(L.GE.2)GO TO 50
      WRITE(6,40)((NAME(I,J),J=1,16),MARK(I),I=1,N)
      STOP
      END
```

EXAMPLE 4 CHANGES THROUGHOUT FILE - 'LOOP', 'RESTART' AND 'TL'

COMMANDS GIVEN

```
BEGIN
   PCCMM;   PNEW
  LOOP
     FC/J/;   RSE?J?JJ?  ;M
*
  RESTART
  TL&
LOOP
  FC (MARK(
   RSE)MARK)NN)
&
   END
```

MESSAGES DURING EXECUTION

```
CCC      PNEW

CCC     LOOP

CCC        FC/J/

CCC      RSE?J?JJ?
NNN         READ(5,30)((NAME(I,JJ),JJ=1,16),MARK(I),I=1,N)

CCC  M
```

EXAMPLE 4 CHANGES THROUGHOUT FILE - 'LOOP', 'RESTART' AND 'TL'

```
CCC        FC/J/

CCC      RSE?J?JJ?
NNN         WRITE(6,40)((NAME(I,JJ),JJ=1,16),MARK(I),I=1,N)

CCC  M

CCC        FC/J/

CCC      RSE?J?JJ?
NNN         DO 70 JJ=1,16

CCC  M

CCC        FC/J/

CCC      RSE?J?JJ?
NNN         MTEMP(JJ)=NAME(I,JJ)

CCC  M

CCC        FC/J/

CCC      RSE?J?JJ?
NNN         NAME(I,JJ)=NAME(I+1,JJ)

CCC  M

CCC        FC/J/
```

EXAMPLE 4 CHANGES THROUGHOUT FILE - 'LOOP', 'RESTART' AND 'TL'

```
CCC     RSE?J?JJ?
NNN     70 NAME(I+1,JJ)=MTEMP(JJ)

CCC  M

CCC       FC/J/

CCC     RSE?J?JJ?
NNN        WRITE(6,40)((NAME(I,JJ),JJ=1,16),MARK(I),I=1,N)

CCC  M

CCC       FC/J/

XXX  PARAMETER OF FOLLOWING COMMAND NOT FOUND FROM LINE NO.    23 TO
XXX  END OF FILE:
XXX       FC/J/
XXX  LAST LINE NOW CURRENT. INTERVENING LINES COPIED TO NEW FILE.

CCC    RESTART

CCC    TL&

CCC  LOOP

CCC    FC (MARK(

CCC     RSE)MARK)NN)
```

EXAMPLE 4. CHANGES THROUGHOUT FILE - 'LOOP', 'RESTART' AND 'TL'

```
NNN         DIMENSION NAME(100,16),NN(100),MTEMP(16)

CCC     FC (MARK(

CCC      RSE)MARK)NN)
NNN         READ(5,30)((NAME(I,JJ),JJ=1,16),NN(I),I=1,N)

CCC     FC (MARK(

CCC      RSE)MARK)NN)
NNN         WRITE(6,40)((NAME(I,JJ),JJ=1,16),NN(I),I=1,N)

CCC     FC (MARK(

CCC      RSE)MARK)NN)
NNN      60 IF(NN(I).GE.NN(I+1))GO TO 80

CCC     FC (MARK(

CCC      RSE)MARK)NN)
NNN         M=NN(I)

CCC     FC (MARK(

CCC      RSE)MARK)NN)
NNN         NN(I)=NN(I+1)
```

EXAMPLE 4 CHANGES THROUGHOUT FILE - 'LOOP', 'RESTART' AND 'TL'

```
CCC     FC (MARK(

CCC      RSE)MARK)NN)
NNN         NN(I+1)=M

CCC     FC (MARK(

CCC      RSE)MARK)NN)
NNN         WRITE(6,40)((NAME(I,JJ),JJ=1,16),NN(I),I=1,N)

CCC     FC (MARK(

XXX  PARAMETER OF FOLLOWING COMMAND NOT FOUND FROM LINE NO.    22 TO
XXX  END OF FILE:
XXX    FC (MARK(
XXX  LAST LINE NOW CURRENT. INTERVENING LINES COPIED TO NEW FILE.

CCC      END

XXX  LENGTH OF AMENDED FILE IS    24 LINES.
```

EXAMPLE 4 CHANGES THROUGHOUT FILE - 'LCOP', 'RESTART' AND 'TL'

AMENDED TEXT

```
      DIMENSION NAME(100,16),NN(100),MTEMP(16)
      READ(5,10)N
   10 FORMAT(I6)
      READ(5,30)((NAME(I,JJ),JJ=1,16),NN(I),I=1,N)
   30 FORMAT(16A4,I6)
      WRITE(6,40)((NAME(I,JJ),JJ=1,16),NN(I),I=1,N)
   40 FORMAT(' ',16A4,I6)
      L=N-1
   50 I=1
   60 IF(NN(I).GE.NN(I+1))GO TO 80
      M=NN(I)
      NN(I)=NN(I+1)
      NN(I+1)=M
      DO 70 JJ=1,16
      MTEMP(JJ)=NAME(I,JJ)
      NAME(I,JJ)=NAME(I+1,JJ)
   70 NAME(I+1,JJ)=MTEMP(JJ)
   80 I=I+1
      IF(I.LE.L)GO TO 60
      L=L-1
      IF(L.GE.2)GO TO 50
      WRITE(6,40)((NAME(I,JJ),JJ=1,16),NN(I),I=1,N)
      STOP
      END
```

EXAMPLE 5 'LOOP N' IS USED TO MAKE AN INSERTION N TIMES AT REGULAR INTERVALS IN FILE. 'PCOMM' ILLUSTRATES THE PROGRESS OF EDITING.

ORIGINAL TEXT

```
THERE'S NOUGHT BUT CARE ON EV'RY HAN',
 IN EVERY HOUR THAT PASSES, O.
WHAT SIGNIFIES THE LIFE O' MAN,
 AN' 'TWERE NA FOR THE LASSES, O.
FOR YOU SAE DOUCE, YE SNEER AT THIS,
 YE'RE NOUGHT BUT SENSELESS ASSES, O.
THE WISEST MAN THE WARL' E'ER SAW,
 HE DEARLY LOV'D THE LASSES, O.
AULD NATURE SWEARS, THE LOVELY DEARS
 HER NOBLEST WORK SHE CLASSES, O.
HER PRENTICE HAN' SHE TRY'D ON MAN,
 AN' THEN SHE MADE THE LASSES, O.
```

COMMANDS GIVEN

```
BEGIN
   PCOMM
    LOOP 3
     M3;   IA
        GREEN GROW THE RASHES, O.
       GREEN GROW THE RASHES, O.
       THE SWEETEST HOURS THAT E'ER I SPEND,
        ARE SPENT AMONG THE LASSES, O.

$
*
 END
```

EXAMPLE 5 'LCOP N'

MESSAGES DURING EXECUTION

```
CCC      LCOP 3

CCC       M3

CCC     IA

CCC       M3

CCC     IA

CCC       M3

CCC     IA

XXX  END OF FILE REACHED. THE ONLY COMMANDS WHICH CAN NOW BE OBEYED
XXX  ARE: END, RESTART, RESET, RETURN, ERASE.

CCC   END

XXX  LENGTH OF AMENDED FILE IS    27 LINES
```

AMENDED TEXT

THERE'S NOUGHT BUT CARE ON EV'RY HAN',
IN EVERY HOUR THAT PASSES, O.
WHAT SIGNIFIES THE LIFE O' MAN,
AN' 'TWERE NA FOR THE LASSES, O.
GREEN GROW THE RASHES, O.
GREEN GROW THE RASHES, O.
THE SWEETEST HOURS THAT E'ER I SPEND,
ARE SPENT AMONG THE LASSES, O.

FOR YOU SAE DOUCE, YE SNEER AT THIS,
YE'RE NOUGHT BUT SENSELESS ASSES, O.
THE WISEST MAN THE WARL' E'ER SAW,
HE DEARLY LOV'D THE LASSES, O.
GREEN GROW THE RASHES, O.
GREEN GROW THE RASHES, O.
THE SWEETEST HOURS THAT E'ER I SPEND,
ARE SPENT AMONG THE LASSES, O.

AULD NATURE SWEARS, THE LOVELY DEARS
HER NOBLEST WORK SHE CLASSES, O.
HER PRENTICE HAN' SHE TRY'D ON MAN,
AN' THEN SHE MADE THE LASSES, O.
GREEN GROW THE RASHES, O.
GREEN GROW THE RASHES, O.
THE SWEETEST HOURS THAT E'ER I SPEND,
ARE SPENT AMONG THE LASSES, O.

EXAMPLE 6 'RETURN' IS USED TO OBTAIN THE SAME RESULT AS IN EXAMPLE 5. 'PCOMM' AND 'PRINT' ILLUSTRATE THE PROGRESS OF EDITING.

ORIGINAL TEXT

GREEN GROW THE RASHES, O.
GREEN GROW THE RASHES, O.
THE SWEETEST HOURS THAT E'ER I SPEND,
ARE SPENT AMONG THE LASSES, O.

THERE'S NOUGHT BUT CARE ON EV'RY HAN',
IN EVERY HOUR THAT PASSES, O.
WHAT SIGNIFIES THE LIFE O' MAN,
AN' 'TWERE NA FOR THE LASSES, O.
FOR YOU SAE DOUCE, YE SNEER AT THIS,
YE'RE NOUGHT BUT SENSELESS ASSES, O.
THE WISEST MAN THE WARL' E'ER SAW,
HE DEARLY LOV'D THE LASSES, O.
AULD NATURE SWEARS, THE LOVELY DEARS
HER NOBLEST WORK SHE CLASSES, O.
HER PRENTICE HAN' SHE TRY'D ON MAN,
AN' THEN SHE MADE THE LASSES, O.

EXAMPLE 6 'RETURN'

COMMANDS GIVEN

```
BEGIN
PCOMM;     PRINT
  D+4;    F10;     RETURN;    F6
      D+3;     F14;    RETURN;     F6
    D+7;   FL;   M;    RETURN;      F6
    DL;    D;    END
```

MESSAGES DURING EXECUTION

```
CCC       PRINT

CCC     D+4

CCC       F10
FFF         THERE'S NOUGHT BUT CARE ON EV'RY HAN',
FFF          IN EVERY HOUR THAT PASSES, O.
FFF         WHAT SIGNIFIES THE LIFE O' MAN,
FFF          AN' 'TWERE NA FOR THE LASSES, O.

CCC        RETURN

CCC       F6
```

EXAMPLE 6 'RETURN'

```
FFF          GREEN GROW THE RASHES, O.
FFF         GREEN GROW THE RASHES, O.
FFF         THE SWEETEST HOURS THAT E'ER I SPEND,
FFF          ARE SPENT AMONG THE LASSES, O.
FFF

CCC        D+3

CCC       F14
FFF        FOR YOU SAE DOUCE, YE SNEER AT THIS,
FFF         YE'RE NOUGHT BUT SENSELESS ASSES, O.
FFF        THE WISEST MAN THE WARL' E'ER SAW,
FFF         HE DEARLY LOV'D THE LASSES, O.

CCC      RETURN

CCC       F6
FFF          GREEN GROW THE RASHES, O.
FFF         GREEN GROW THE RASHES, O.
FFF         THE SWEETEST HOURS THAT E'ER I SPEND,
FFF          ARE SPENT AMONG THE LASSES, O.
FFF

CCC       D+7

CCC      FL
```

EXAMPLE 6 'RETURN'

```
FFF       AULD NATURE SWEARS, THE LOVELY DEARS
FFF        HER NOBLEST WORK SHE CLASSES, O.
FFF       HER PRENTICE HAN' SHE TRY'D ON MAN,

CCC    M
FFF        AN' THEN SHE MADE THE LASSES, O.

XXX  END OF FILE REACHED. THE ONLY COMMANDS WHICH CAN NOW BE OBEYED
XXX  ARE: END, RESTART, RESET, RETURN, ERASE.

CCC     RETURN

CCC      F6
FFF          GREEN GROW THE RASHES, O.
FFF         GREEN GROW THE RASHES, O.
FFF         THE SWEETEST HOURS THAT E'ER I SPEND,
FFF          ARE SPENT AMONG THE LASSES, O.
FFF

CCC      DL

CCC     D

XXX  END OF FILE REACHED. THE ONLY COMMANDS WHICH CAN NOW BE OBEYED
XXX  ARE: END, RESTART, RESET, RETURN, ERASE.

CCC     END

XXX  LENGTH OF AMENDED FILE IS    27 LINES.
```

EXAMPLE 6 'RETURN'

AMENDED TEXT

THERE'S NOUGHT BUT CARE ON EV'RY HAN',
IN EVERY HOUR THAT PASSES, O.
WHAT SIGNIFIES THE LIFE O' MAN,
AN' 'TWERE NA FOR THE LASSES, O.
GREEN GROW THE RASHES, O.
GREEN GROW THE RASHES, O.
THE SWEETEST HOURS THAT E'ER I SPEND,
ARE SPENT AMONG THE LASSES, O.

FOR YOU SAE DOUCE, YE SNEER AT THIS,
YE'RE NOUGHT BUT SENSELESS ASSES, O.
THE WISEST MAN THE WARL' E'ER SAW,
HE DEARLY LOV'D THE LASSES, O.
GREEN GROW THE RASHES, O.
GREEN GROW THE RASHES, O.
THE SWEETEST HOURS THAT E'ER I SPEND,
ARE SPENT AMONG THE LASSES, O.

AULD NATURE SWEARS, THE LOVELY DEARS
HER NOBLEST WORK SHE CLASSES, O.
HER PRENTICE HAN' SHE TRY'D ON MAN,
AN' THEN SHE MADE THE LASSES, O.
GREEN GROW THE RASHES, O.
GREEN GROW THE RASHES, O.
THE SWEETEST HOURS THAT E'ER I SPEND,
ARE SPENT AMONG THE LASSES, O.

EXAMPLE 7 THIS DEMONSTRATES THE CUMULATION OF ERRORS THAT MAY RESULT IF THE 'EXIT' COMMAND IS NOT IN FORCE. IT IS ASSUMED THAT THE COMMAND 'FC/GE/' HAS BEEN GIVEN IN MISTAKE FOR 'FC/GT/'. 'PCOMM' ILLUSTRATES THE PROGRESS OF EDITING.

ORIGINAL TEXT

```
   DIMENSION NAME(100,16),MARK(100),MTEMP(16)
   READ(5,10)N
10 FORMAT(I6)
   READ(5,30)((NAME(I,J),J=1,16),MARK(I),I=1,N)
30 FORMAT(16A4,I6)
   WRITE(6,40)((NAME(I,J),J=1,16),MARK(I),I=1,N)
40 FORMAT(' ',16A4,I6)
   L=N-1
60 IF(MARK(I).GT.MARK(I+1))GO TO 80
   M=MARK(I)
   MARK(I)=MARK(I+1)
   MARK(I+1)=M
   DO 70 J=1,16
   MEMP(J)=NAME(I,J)
   NAME(I,J)=NAME(I+1,J)
70 NAME(I+1,J)=MTEMP(J)
80 I=I+1
   IF(I.LE.L)GO TO 60
   L=L-1
   IF(L.GE.2)GO TO 50
   WRITE(6,40)((NAME(I,J),J=1,16),MARK(I),I=1,N)
   STOP
   END
```

EXAMPLE 7 CUMULATION OF ERRORS WITHOUT 'EXIT'

COMMANDS GIVEN

```
BEGIN
PCOMM
FC/GE/;  P ;  RS/T/E/;  P;  IB
   50 I=1
£
P;  FB/ME/; P; ISA/M/T/; P; END
```

MESSAGES DURING EXECUTION

```
 CCC  FC/GE/

 CCC    P
       IF(L.GE.2)GO TO 50
 XXX  CURRENT LINE IS NO.    20. NO. OF LINES IN FILE IS    23

 CCC    RS/T/E/

 CCC    P
       IF(L.GE.2)GO EO 50
 XXX  CURRENT LINE IS NO.    20. NO. OF LINES IN FILE IS    23

 CCC    IB
```

EXAMPLE 7 CUMULATION OF ERRORS WITHOUT 'EXIT'

```
CCC  P
      IF(L.GE.2)GO EO 50
XXX  CURRENT LINE IS NO.    20. NO. OF LINES IN FILE IS    23

CCC    FB/ME/

XXX  PARAMETER OF FOLLOWING COMMAND NOT FOUND FROM LINE NO.    20 TO
XXX  END OF FILE:
XXX    FB/ME/
XXX  LAST LINE NOW CURRENT. INTERVENING LINES COPIED TO NEW FILE.

CCC   P
      END
XXX  CURRENT LINE IS NO.    23. NO. OF LINES IN FILE IS    23

CCC   ISA/M/T/

XXX  PARAMETER(S) OF FOLLOWING COMMAND NOT FOUND IN LINE SPECIFIED:
XXX   ISA/M/T/
XXX  NO ACTION TAKEN.

CCC   P
      END
XXX  CURRENT LINE IS NO.    23. NO. OF LINES IN FILE IS    23

CCC   END

XXX  LENGTH OF AMENDED FILE IS    24 LINES.
```

EXAMPLE 7 CUMULATION OF ERRORS WITHOUT 'EXIT'

AMENDED TEXT

```
   DIMENSION NAME(100,16),MARK(100),MTEMP(16)
   READ(5,10)N
10 FORMAT(I6)
   READ(5,30)((NAME(I,J),J=1,16),MARK(I),I=1,N)
30 FORMAT(16A4,I6)
   WRITE(6,40)((NAME(I,J),J=1,16),MARK(I),I=1,N)
40 FORMAT(' ',16A4,I6)
   L=N-1
60 IF(MARK(I).GT.MARK(I+1))GO TO 80
   M=MARK(I)
   MARK(I)=MARK(I+1)
   MARK(I+1)=M
   DO 70 J=1,16
   MEMP(J)=NAME(I,J)
   NAME(I,J)=NAME(I+1,J)
70 NAME(I+1,J)=MTEMP(J)
80 I=I+1
   IF(I.LE.L)GO TO 60
   L=L-1
50 I=1
   IF(L.GE.2)GO EO 50
   WRITE(6,40)((NAME(I,J),J=1,16),MARK(I),I=1,N)
   STOP
   END
```

EXAMPLE 7A AS EXAMPLE 7, WITH 'EXIT' COMMAND INSERTED.
IN ON-LINE EDITING, 'ERASE' OR 'RESET 3' WOULD BE GIVEN TO
RECOVER FROM THE ERROR.

ORIGINAL TEXT AS IN EXAMPLE 7

COMMANDS GIVEN

```
BEGIN
PCOMM
  EXIT
FC/GE/;  P ;  RS/T/E/;  P;  IB
   50 I=1
£
P;  FB/ME/; P; ISA/M/T/; P; END
```

EXAMPLE 7A 'EXIT'

MESSAGES DURING EXECUTION

```
CCC    EXIT

CCC  FC/GE/

CCC    P
       IF(L.GE.2)GO TO 50
XXX  CURRENT LINE IS NO.     20. NO. OF LINES IN FILE IS     23

CCC    RS/T/E/

CCC    P
       IF(L.GE.2)GO EO 50
XXX  CURRENT LINE IS NO.     20. NO. OF LINES IN FILE IS     23

CCC    IB

CCC  P
       IF(L.GE.2)GO EC 50
XXX  CURRENT LINE IS NO.     20. NO. OF LINES IN FILE IS     23

CCC    FB/ME/

XXX  PARAMETER OF FOLLOWING COMMAND NOT FOUND FROM LINE NO.     20 TO
XXX  END OF FILE:
XXX    FB/ME/
XXX  LAST LINE NOW CURRENT. INTERVENING LINES COPIED TO NEW FILE.
XXX  EXECUTION SUSPENDED BY EXIT COMMAND.

XXX  LENGTH OF AMENDED FILE IS     24 LINES.
```

EXAMPLE 7A 'EXIT'

AMENDED TEXT

```
   DIMENSION NAME(100,16),MARK(100),MTEMP(16)
   READ(5,10)N
10 FORMAT(I6)
   READ(5,30)((NAME(I,J),J=1,16),MARK(I),I=1,N)
30 FORMAT(16A4,I6)
   WRITE(6,40)((NAME(I,J),J=1,16),MARK(I),I=1,N)
40 FORMAT(' ',16A4,I6)
   L=N-1
60 IF(MARK(I).GT.MARK(I+1))GO TO 80
   M=MARK(I)
   MARK(I)=MARK(I+1)
   MARK(I+1)=M
   DO 70 J=1,16
   MEMP(J)=NAME(I,J)
   NAME(I,J)=NAME(I+1,J)
70 NAME(I+1,J)=MTEMP(J)
80 I=I+1
   IF(I.LE.L)GO TO 60
   L=L-1
50 I=1
   IF(L.GE.2)GO EO 50
   WRITE(6,40)((NAME(I,J),J=1,16),MARK(I),I=1,N)
   STOP
   END
```

EXAMPLE 7B THE USE OF 'EXIT' WITH A PARAMETERLESS 'LOOP' IN BACKGROUND EDITING MAY BE INADVISABLE IF FURTHER COMMANDS FOLLOW THE LOOP. COMPARE THE RESULTS OF THIS EXAMPLE WITH 'EXIT' WITH THOSE OF EXAMPLE 4 WITHOUT IT.

ORIGINAL TEXT AS IN EXAMPLE 4

COMMANDS GIVEN

```
BEGIN
   PCOMM;   PNEW
EXIT
  LOOP
     FC/J/;   RSE?J?JJ?  ;M
*
  RESTART
  TL&
LOOP
  FC (MARK(
   RSE)MARK)NN)
&
   END
```

MESSAGES DURING EXECUTION

```
CCC     PNEW

CCC  EXIT

CCC    LOOP
```

EXAMPLE 7B 'EXIT' SOMETIMES INADVISABLE WITH 'LOOP'

```
CCC        FC/J/

CCC      RSE?J?JJ?
NNN         READ(5,30)((NAME(I,JJ),JJ=1,16),MARK(I),I=1,N)

CCC   M

CCC        FC/J/

CCC      RSE?J?JJ?
NNN         WRITE(6,40)((NAME(I,JJ),JJ=1,16),MARK(I),I=1,N)

CCC   M

CCC        FC/J/

CCC      RSE?J?JJ?
NNN         DO 70 JJ=1,16

CCC   M

CCC        FC/J/

CCC      RSE?J?JJ?
NNN         MTEMP(JJ)=NAME(I,JJ)

CCC   M

CCC        FC/J/
```

EXAMPLE 7B 'EXIT' SOMETIMES INADVISABLE WITH 'LOOP'

```
CCC      RSE?J?JJ?
NNN         NAME(I,JJ)=NAME(I+1,JJ)

CCC   M

CCC         FC/J/

CCC      RSE?J?JJ?
NNN      70 NAME(I+1,JJ)=MTEMP(JJ)

CCC   M

CCC         FC/J/

CCC      RSE?J?JJ?
NNN         WRITE(6,40)((NAME(I,JJ),JJ=1,16),MARK(I),I=1,N)

CCC   M

CCC         FC/J/

XXX   PARAMETER OF FOLLOWING COMMAND NOT FOUND FROM LINE NO.     23 TO
XXX   END OF FILE:
XXX         FC/J/
XXX   LAST LINE NOW CURRENT. INTERVENING LINES COPIED TO NEW FILE.
XXX   EXECUTION SUSPENDED BY EXIT COMMAND.

XXX   LENGTH OF AMENDED FILE IS     24 LINES.
```

EXAMPLE 7B 'EXIT' SOMETIMES INADVISABLE WITH 'LOOP'

AMENDED TEXT

```
   DIMENSION NAME(100,16),MARK(100),MTEMP(16)
   READ(5,10)N
10 FORMAT(I6)
   READ(5,30)((NAME(I,JJ),JJ=1,16),MARK(I),I=1,N)
30 FORMAT(16A4,I6)
   WRITE(6,40)((NAME(I,JJ),JJ=1,16),MARK(I),I=1,N)
40 FORMAT(' ',16A4,I6)
   L=N-1
50 I=1
60 IF(MARK(I).GE.MARK(I+1))GO TO 80
   M=MARK(I)
   MARK(I)=MARK(I+1)
   MARK(I+1)=M
   DO 70 JJ=1,16
   MTEMP(JJ)=NAME(I,JJ)
   NAME(I,JJ)=NAME(I+1,JJ)
70 NAME(I+1,JJ)=MTEMP(JJ)
80 I=I+1
   IF(I.LE.L)GO TO 60
   L=L-1
   IF(L.GE.2)GO TO 50
   WRITE(6,40)((NAME(I,JJ),JJ=1,16),MARK(I),I=1,N)
   STOP
   END
```

EXAMPLE 8 THE 'TI' COMMAND IS NEEDED WHEN '£' OCCURS IN COLUMN 1 OF THE FILE TO BE EDITED, OTHERWISE THE EDITOR WILL TAKE THE '£' TO INDICATE THE END OF THE FILE.

ORIGINAL TEXT

```
BEGIN
    PCOMM;PNEW
P;IB
      INSERTION BEFORE LINE 1
£
   P;    IA;
      INSERTION AFTER LINE 1
£
   P;   IB4
      FIRST INSERTION BEFORE LINE 4
      SECOND INSERTION BEFORE LINE 4
£
   P;    IB5
      FIRST INSERTION AFTER LINE 5
      SECOND INSERTION AFTER LINE 5
      THIRD INSERTION AFTER LINE 5
£
   P;  R
      FIRST LINE TO REPLACE LINE 6
      SECOND LINE TO REPLACE LINE 6
£
   P;    R12/13
      REPLACEMENT FOR LINES 12-13
£
END
```

COMMANDS GIVEN

```
BEGIN;  TI ?
FC/IB5/  ;  RS*B*A*;   END
```

EXAMPLE 8 'TI'

MESSAGES DURING EXECUTION

```
XXX  LENGTH OF AMENDED FILE IS    25 LINES.
```

AMENDED TEXT

```
BEGIN
    PCOMM;PNEW
P;IB
      INSERTION BEFORE LINE 1
£
   P;    IA;
      INSERTION AFTER LINE 1
£
   P;   IB4
      FIRST INSERTION BEFORE LINE 4
      SECOND INSERTION BEFORE LINE 4
£
   P;    IA5
      FIRST INSERTION AFTER LINE 5
      SECOND INSERTION AFTER LINE 5
      THIRD INSERTION AFTER LINE 5
£
   P;  R
      FIRST LINE TO REPLACE LINE 6
      SECOND LINE TO REPLACE LINE 6
£
   P;    R12/13
      REPLACEMENT FOR LINES 12-13
£
END
```

REFERENCES

1. GBEDIT. [Computer listing of a context editor for use from RAX 2260 or CDC terminals. University of St. Andrews Computing Laboratory, 1973]

2. Morrison, R., Howson, E., Sommerville, I. TED: a context editor user's manual. (Technical report no. CL/73/3) University of St Andrews Computing Laboratory, 1973.

3. Poole, P.C. MITEM; a portable program for the manipulation of text. Preliminary user manual. (Draft) Abingdon, Berks.: U.K.A.E.A. Research Group, Culham Laboratory, 1971.

4. van Dam, Andries and Rice, David E. Computers and publishing: writing, editing, and printing. Advances in computers, 10 (1970) 145 - 74.

5. van Dam, Andries and Rice, David E. On-line text editing: a survey. Computing surveys, 3, no. 3 (Sept. 1971) 93 - 114.

6. International Computers Limited. 4100 technical manual. Reading, Berks. (Section 2.13.10 deals with EDIT facilities)

7. International Computers Limited. Compiling systems. (Manual no. 4241) Reading, Berks., 1971. (Chapter 5: Editor programs. pp. 33 - 82)

8. Engelbart, Douglas C. and English, William K. A research center for augmenting human intellect. AFIPS conference proceedings, 33, pt 1. 1968 Fall Joint Computer Conference. 395 - 410.

9. Bozman, William R. Computer-aided typesetting. Advances in computers, 7 (1966) 195 - 207.

10. Martin, David. Computer typesetting and information. Journal of documentation, 28, no. 3 (Sept. 1972) 247 - 61.

11. Phillips, Arthur. Computer peripherals and typesetting. London: HMSO, 1968.

12. Teperman, A. and Katzenelson, J. A format editor. Software - practice and experience, 2 (1972) 219 - 30.

13. Poole, M.D. Implementation of an editing algorithm allowing repeating corrections. Software - practice and experience, 1 (1971) 373 - 81.

14. Deutsch, L. Peter and Lampson, Butler W. An online editor. Communications of the ACM, 10, no. 12 (Dec. 1967) 793 - 9, 803.

15. Brown, P.J. The ML/I macro processor. ML/I user's manual. 4th ed. Canterbury: Computing Laboratory, University of Kent at Canterbury, 1970 (repr. with supplements 1972).

16. Brown, P.J. The ML/I macro processor. Communications of the ACM, 10, no. 10 (Oct. 1967) 618 - 23.

17. Bourne, S.R. A design for a text editor. Software - practice and experience, 1 (1971) 73 - 81.

18. Mooers, Calvin N. TRAC, a procedure-describing language for the reactive typewriter. Communications of the ACM, 9, no.3 (March 1966) 215 - 9.

19. Higman, B. A comparative study of programming languages. London: Macdonald [1967]

20. Strachey, C. A general purpose macrogenerator. Computer journal, 8 (1965/66) 225 - 41.

21. Waite, William M. A language-independent macro processor. Communications of the ACM, 10, no. 7 (July 1967) 433 - 440.

22. Farber, D.J., Griswold, R.E. and Polonsky, I.P. The SNOBOL3 programming language. Bell System technical journal, 45 (1966) 895 - 944.

23. Forte, Allen. SNOBOL3 primer. [Cambridge, Mass.: MIT Press, 1967].

24. Benjamin, Arthur J. An extensible editor for a small machine with disk storage. Communications of the ACM, 15, no. 8 (Aug. 1972) 742 - 7.

25. Barnett, Michael P. Computer programming in English. New York: Harcourt, Brace & World [1969].

26. COMIT programmers' reference manual. Cambridge, Mass.: MIT Research Laboratory of Electronics and the Computation Center [1961].

27. Griswold, R.E., Poage, J.F. and Polonsky, I.P. The SNOBOL4 programming language. 2nd ed. Englewood Cliffs, N.J.: Prentice-Hall [1971].

28. Griswold, Ralph E. The macro implementation of SNOBOL4. San Francisco: W. H. Freeman [1972].

29. Vine, R.E. *Ed; the MTS file editing program. (Northumbrian Universities multiple access computer. Programming note 28.) [Computing Laboratory, University of Newcastle upon Tyne] 1970.

30. Colin, A.J.T. DOLPHIN - a text filing system for university use. Computer journal, 13, no. 2 (May 1970) 136 - 41.

31. A Users' guide to COTAN. [Abingdon, Berks.]: Computing and Applied Mathematics Group, Culham Laboratory, 1968. (COTAN 3 command - AMEND, pp. 36 - 46).

32. Brown, H. EDITDC: a sub-system for editing disc files. (Kent On-line System. Document: KUSE/EDITDC/1.) University of Kent at Canterbury, 1970.

33. University of Bradford. Computing Laboratory. Specification of the Bradford Program Editing and Storage System. BESS Mk. 7. 1971. (Input - the amendments. pp. 16 - 17)

34. University College of Wales, Aberystwyth. Computer Unit. Disc-file editor. 1971.

35. University of Strathclyde. Department of Computer Science. [Strathclyde Operating System manual. 1971] (Chapter 3: The UNICORN file store. The 'EDIT' command. pp. 8 - 12).

36. University of Southampton. Computing Service. MINIMOP editor, Mark II. [1970].

37. Control Data Corporation. 6000 series computer systems. Intercom2 reference manual. [Sunnyvale, Calif., 1970] (Chapter 4: SETUP)

38. Moudrý, J. A notation describing corrections in files. Software - practice and experience, 2 (1972) 279 - 85.

39. University Mathematical Laboratory, Cambridge. Titan library routine specification. Complete program EDIT. 1967.

40. University of Manchester. Department of Computer Science. MU/MX Operating System. System manual. 1971. (Chapter 4.4: The text file editor. pp. 51 - 73).

Although not mentioned in the text, the following have been found relevant:

Chapter 1

Danielson, Wayne A. The man-machine combination for computer-assisted copy editing. Advances in computers, 7 (1966) 181 - 93.

Kunkel, G.Z. and Marcum, T.H. Hyphenless justification. Datamation, 11, no. 4 (1965) 42.

Martins, Gary R. Dimensions of text processing. AFIPS conference proceedings, 41, pt 2. 1972 Fall Joint Computer Conference. 801 - 10.

Rich, R.P. and Stone, A.G. Method for hyphenating at the end of a printed line. Communications of the ACM, 8, no. 7 (July 1965) 444 - 5.

Chapter 2

Bobrow, D.G. and Raphael, B. A comparison of list-processing computer languages. Communications of the ACM, 7, no. 4 (April 1964) 231 - 40.

Raphael, B. Aspects and applications of symbol manpulation [sic]. In Proceedings of 21st National Conference, Association for Computing Machinery. [London]: Academic Press, 1967. pp. 69 - 74.

Sammet, Jean E. Programming languages: history and fundamentals. Englewood Cliffs, N.J.: Prentice-Hall [1969]

Wegner, P. Programming languages, information structures, and machine organization. New York: McGraw-Hill [1968].

Chapter 3

Bratman, H. and others. Program composition and editing with an on-line display. AFIPS conference proceedings, 33, pt 2. 1968 Fall Joint Computer Conference. 1349 - 60.

Irons, Edgar T. and Djorup, Frans M. A CRT editing system. Communications of the ACM, 15, no. 1 (Jan. 1972) 16 - 20.

Wells, M. and others. Eldon 2 manual. ... Designed and implemented by the staff of the Electronic Computing Laboratory at the University of Leeds. 1969. (Section 2.2: Amendment facilities. [5p.])

Wilkes, M.A. Conversational access to a 2048-word machine. Communications of the ACM, 13, no. 7 (July 1970) 407 - 14.

APPENDIX A

E D I T

A Program for Amending Text Files

User's Guide

C O N T E N T S

INTRODUCTION

EDIT may be used on-line under RAX or in background jobs under RAX or 44MFT.

General Features of EDIT

File to be Edited

1. The file to be edited is (a) on-line, a SYSLIB file previously saved by a /SAVE command (b) in background jobs, a RAX SYSFIL file or a 44MFT disk or tape dataset.
(Other input files are possible - see Note 8.)
(Warning - If a pound sign (£) occurs in column 1 of any line of the file, see the TI command.)

2. Editing is carried out on a copy of the file, so the original remains unchanged.

Edited File

3. During editing, a new file is created, with the necessary amendments introduced.

4. At the end of the editing session, the edited file is
(a) on-line, one that may be saved by /SAVE name(lock),SV
(b) in background use, one written to a disk or tape dataset.
(Other output files are possible - see Note 8.)

Line Length

5. Each line of the file is assumed to be 80 characters in length.
(Other lengths are possible - see Note 8.)

Commands

6. EDIT has a language of editing commands which specify the operations to be performed on the file. These commands (and any text lines for insertion - but see also Note 8) are read from the console in on-line editing and from the card-reader in background. (Other command files are possible - see Note 8.) Commands are obeyed interpretively, if one command fails, EDIT attempts to execute the next one (but see the EXIT command).

Current Line

7. During editing, the file is read sequentially from the first line to the last. As each line is read, it becomes the current one - the one being "pointed at". At the start of an editing session, the first line is current. Editing commands may refer to the current line or may move the file forward so that a later line becomes current. No command may refer to a point in the file earlier than the current line, although a few, special "Control" commands do move the file backwards.

Options

8. EDIT provides "Option" commands which may be used to alter the text line length or the units for reading or writing certain files, when the normal procedure is unsuitable.

SUMMARY AND INDEX OF COMMANDS

		Page
LINE n	Make n (≤ 132) the number of characters in each line of text.	25
UI n	Read input for editing from unit n.	25
UO n	Write edited output on unit n.	25
UC n	Read commands from unit n.	25
UA n	Read added lines for insertion from unit n.	25
TI char	Make "char" the terminator of input file for editing.	25
TA char	Make "char" the terminator of added lines.	25
TL char	Make "char" the terminator of a LOOP sequence.	25

TYPING OF COMMANDS

General

Spacing is not important- except within character string parameters.

Commands may be entered either one at a time or several to a line, using the semi-colon as a separator.

Example:

either

```
F 12
P
DS /AND/
```

or

```
F12; P ; DS/AND/
```

Note. The only commands which can follow on the same line as the BEGIN command are "Option" ones.

(Warning. There is the danger of an accumulation of errors if several commands are entered and executed together - but see EXIT, ERASE and RESET commands.)

A command must not be divided between the end of one line and the beginning of the next unless it is one with very long character string parameters (see below).

Text for insertion following IA, IB and R commands and lines in a LOOP sequence must start on the next line and not on the same line as the command itself.

Example:

```
F8; ISB /CHANCE/HAPPY /; LOOP 4
   FC /ONTO/ ; RS /ONTO/ON TO/
*
```

not

```
F8; ISB /CHANCE/HAPPY /; LOOP 4; FC /ONTO/
RS /ONTO/ON TO/
*
```

Integer Parameters

These are referred to as "m" and "n" in the description of commands. They must be positive and unsigned.

Character String Parameters

These are referred to as "string", "string1" and "string2" in the description of commands.

They may be of any length up to the length of the text line, but should be as short as possible.

The delimiters of a string - represented by the solidus (/) in the definitions of the commands - may be any character (except blank or E or S) not contained in the particular string.

All blanks in character strings are significant except leading blanks in FB and DB commands. (This exception does away with the need to count and type initial blanks.)

If a character string is so long that typing has to continue on a new line, a continuation character (+) must be typed in column 1 of that line.

```
Example:  ISA /UNDERSTOOD. /THIS IS AN UNUSUALLY LONG ST
         +RING WHICH FOR SOME REASON HAS TO BE INSERTED./
```

FIND and MOVE commands

These copy up to, but not including, the specified line, which becomes the current one.

F n Find line no. n.

For example:
```
F24
F 229;
```

FL — Find last line.

This is useful if an insertion is wanted at the end of the file and the number of the last line is unknown.

FB /string/ — Find line beginning with "string".

Leading blanks in the line and in "string" are ignored.

For example: The Fortran statement

```
20 II=II+1
```

would be identified by either of the following

```
FB *20*
FB &        20&
```

but not by

```
FB ?20    ?
```

FC /string/ — Find line containing "string".

All blanks are significant.

For example: The line

```
IF(X.EQ.Y)GO TO 65
```

would be identified by either of the following

```
FC /GO TO/
FC *GO TO *
```

but not by either of the following

```
FC / GO TO/
FC *GOTO*
```

Note: FB is more efficient than FC and should be used when possible.

M [n] Move n lines beyond current one.

If n is omitted, it is assumed to be 1.

For example: M
M13;

DELETE commands

Deletion of complete lines

D [+n] Delete current line [and following n lines].

The line after the last deleted one becomes current.

For example: D;
D+4

DA n Delete from current line to after line no. n.

Line no. n+1 becomes current.

For example: DA56

D n Delete line no. n.

Line no. n+1 becomes current.

For example: D8;

D m/n Delete from line no. m (≤n) to line no. n inclusive.

Line no. n + 1 becomes current.

For example: D 6/12

DL Delete from current line to before last line.

Last line becomes current.

For example: DL; D

This deletes from current line to last line, inclusive.

DB /string/ Delete from current line to before line beginning with "string" - which becomes current.

See notes under FB.

For example: DB /THE NEXT/;D

This deletes to after line beginning "THE NEXT".

DC /string/ Delete from current line to before line containing "string" - which becomes current.

See notes under FC.

For example: DC *UP TO*; D

This deletes to after line containing "UP TO".

Deletion within a line

DS /string/ Delete first occurrence of "string" in current line.

DSE /string/ Delete every occurrence of "string" in current line.

DSS /string1/string2/ Delete from first occurrence of "string1" to first following occurrence of "string2", inclusive, in current line.

This saves the typing of a long string in a DS command.

After the execution of any of these commands, the same line remains current unless all non-blank characters have been removed, in which case the line is considered deleted and the next line becomes current.

Example 1: Following the command

DS / VERY/

the line

IT IS VERY STRANGE AND VERY TRUE.

becomes

IT IS STRANGE AND VERY TRUE.

If the command is repeated, the line becomes

IT IS STRANGE AND TRUE.

Example 2: The same result can be achieved more efficiently by the single command

```
DSE / VERY/
```

Example 3: Following the command

```
DSS /,0/5)/
```

the line

```
DIMENSION OLD(600,10),OLDER(300),OLDEST(100),YOUNG(5),YOUNGER(10)
```

becomes

```
DIMENSION OLD(600,10),YOUNGER(10)
```

INSERT commands

Insertion of complete lines

IA [n] Insert line(s) following IA after line no. n. Line no. n+1 becomes current.

IB [n] Insert line(s) following IB before line no. n. Line no. n remains current.

If n is omitted, it is assumed to be the number of the current line.

The first line for insertion must not be typed on the same line as the command.

The end of the insertion must be indicated by a terminating character (default option £) in column 1 on a line by itself.

Example 1: Suppose the following lines are part of a file, with "Z=A/B" the current line

```
    X=Y+Z
    Z=A/B  [current line]
    GO TO 100
 45 IF(KOUNT.LE.4)GO TO 30
```

then the command

```
IA
   WRITE(6,20)X
   WRITE(6,20)Y
£
```

produces

```
X=Y+Z
Z=A/B
WRITE(6,20)X
WRITE(6,20)Y
GO TO 100     [new current line]
45 IF(KOUNT.LE.4)GO TO 30
```

Example 2: Suppose that in the original sequence of lines, "Z=A/B" is line no. 8, then the command

```
IB 8
    WRITE(6,20)X
    WRITE(6,20)Y
 £
```

produces

```
X=Y+Z
WRITE(6,20)X
WRITE(6,20)Y
Z=A/B        [this remains the current line]
GO TO 100
45 IF(KOUNT.LE.4)GO TO 30
```

Insertion within a line

ISA /string1/string2/ Insert "string2" after first occurrence of "string1" in current line.

ISAE /string1/string2/ Insert "string2" after every occurrence of "string1" in current line.

ISB /string1/string2/ Insert "string2" before first occurrence of "string1" in current line.

ISBE /string1/string2/ Insert "string2" before every occurrence of "string1" in current line.

Insertion causes the loss of space characters at the end of the line. If any non-blank characters overflow, a new line is created to hold them and it is considered the current line. Normally the same line remains current.

Example 1: Following the command

```
ISA *VERY*, VERY*
```

the line

```
IT IS VERY STRANGE AND VERY TRUE.
```

becomes

```
IT IS VERY, VERY STRANGE AND VERY TRUE.
```

If the command is repeated, the line becomes

```
IT IS VERY, VERY, VERY STRANGE AND VERY TRUE.
```

Example 2: Following the command

```
ISAE *VERY*, VERY*
```

the original line above becomes

```
IT IS VERY, VERY STRANGE AND VERY, VERY TRUE.
```

Example 3: If the following line is current

```
IF((K.EQ.L).OR.(K.EQ.LL))K=K+1
```

then the sequence of commands

```
ISA /)/.OR.(K.NE.II)/
ISBE &K&J&
```

produces

```
IF((JK.EQ.L).OR.(JK.NE.II).OR.(JK.EQ.LL))JK=JK+1
```

REPLACE commands

Replacement of complete lines

R [+ n] Replace current line [and following n lines] with line(s) following command.

R = Replace an equal number of lines, from the current one onwards, with the line(s) following the command.

R n Replace line no. n with line(s) following command.

R m/n Replace lines no. m (≤n) to n inclusive with line(s) following command.

The line following the replaced line(s) becomes current.

The end of the insertion must be indicated by a terminating character (default option £) in column 1 on a line by itself.

Examples: Suppose the following lines are part of a file, with "Z=A/B" the current line.

```
   X=Y+Z
   Z=A/B      [current line]
   GO TO 100
45 IF(KOUNT.LE.4)GO TO 30
   IF(K.EQ.L) RETURN
```

(1)
```
   R
   WRITE(6,20)X
   WRITE(6,20)Y
£
```
produces
```
   X=Y+Z
   WRITE(6,20)X
   WRITE(6,20)Y
   GO TO 100    [new current line]
45 IF(KOUNT.LE.4)GO TO 30
   IF(K.EQ.L) RETURN
```

(2)
```
R + 2
   WRITE(6,20)X
   WRITE(6,20)Y
£
```
produces
```
   X=Y+Z
   WRITE(6,20)X
   WRITE(6,20)Y
   IF(K.EQ.L) RETURN [new current]
```

(3)
```
R =
   WRITE(6,20)X
   WRITE(6,20)Y
£
```
produces
```
   X=Y+Z
   WRITE(6,20)X
   WRITE(6,20)Y
45 IF(KOUNT.LE.4)GO TO 30  [new current]
   IF(K.EQ.L) RETURN
```

Replacement within a line

RS /string1/string2/ Replace first occurrence of "string1" in current line by "string2".

RSE /string1/string2/ Replace every occurrence of "string1" in current line by "string2".

If "string2" is longer than "string1", space characters will be lost at the end of the line. If any non-blank characters overflow, a new line is created to hold them and it is considered the current line. Normally the same line remains current.

Example 1: If the following line is current

```
IF((K.EQ.L).OR.(K.EQ.LL))K=K+1
```

the command

```
RS /K/M/
```

produces

```
IF((M.EQ.L).OR.(K.EQ.LL))K=K+1
```

Example 2: The command

```
RSE &K&M&
```

amends the line above to

```
IF((M.EQ.L).OR.(M.EQ.LL))M=M+1
```

LOOP command

LOOP [n] Execute n times the sequence of commands following LOOP. If n is omitted, repetition continues to the end of the file or until a command can no longer be obeyed.

A maximum of ten lines may follow LOOP, followed by a terminating character (default option *) in column 1 on a line by itself.

The first line of the sequence must not be typed on the same line as LOOP.

Another LOOP command must not appear in a LOOP sequence.

Since the action of the parameterless LOOP command must involve movement up to, or towards, the end of the file, the RESTART and RETURN commands cannot occur in the sequence. They may appear in a LOOP n sequence.

Example 1:
```
LOOP 3
 M4 ; D
*
```

This sequence deletes the fourth, ninth and fourteenth lines beyond the current one.

Example 2:
```
LOOP
 M4 ; D
*
```

This deletes every fifth line until the end of file is reached or until the M4 command cannot be obeyed.

Example 3:
```
LOOP 6
DS /X/
*
```

This deletes the first six occurrences of "X" from the current line.

Example 4:
```
 LOOP 4
 FL; M
RETURN
*
END
```

If the first line of the file is current when this LOOP command is given, the new file will consist of five successive copies of the old.

```
Example 5:      LOOP 100
                 FL; IA
                  XXX
               £
                RESTART
               *
```

This inserts at the end of the file 100 lines, each consisting of "XXX".

(This operation could be performed more efficiently in a programming language and is not recommended under EDIT!)

```
Example 6:      LOOP  6
                 M4; IB
             First inserted line
                . . .
                . . .
             Last inserted line
        £
        *
```

This inserts some lines before every fourth line of the file until the operation has been performed six times.

Note. The number of inserted lines could not be greater than eight, because of the limit on the length of the LOOP sequence.

```
Example 7:      LOOP
               FC *KK*
               RSE *KK*MM*
             *
```

This replaces every occurrence of "KK" by "MM" from the current line to the end of the file.

```
Example 8:      LOOP
               FC *KK*
               RS *KK*MM*
             *
```

This achieves the same result as in Example 7, but is less efficient.

```
Example 9:      LOOP 7
                FC *KK*
                RS *KK*MM*
               *
```

This replaces the next seven occurrences of "KK" by "MM".

```
Example 10:     LOOP 7
                FC *KK*
                RSE *KK*MM*
               *
```

This finds the next seven lines in which "KK" occurs and changes each occurrence to "MM".

```
Example 11:     LOOP 10
                 IB

               £
               *
```

This inserts ten blank lines before the current one.

Control commands

BEGIN Begin the editing session.

This must be given as the first command of the session.

Only OPTION commands can be typed on the same line as BEGIN.

END End the editing session.

This command must be given.

If the end of file has not been reached when it is given, the remainder of the file is copied at this point.

RESTART Make the amended file into the file to be edited.

If the end of file has not been reached when the command is given, the remainder of the file is copied at this point and the first line of the amended file becomes current.

Line numbers now refer to this file.

Example: The following sequence of commands replaces "DUCKS" by "DRAKES" and "GEESE" by "GANDERS" throughout a file

```
LOOP
FC/DUCKS/
RSE/DUCKS/DRAKES/
*
RESTART
LOOP
FC/GEESE/; RSE/GEESE/GANDERS/
*
```

RETURN Return the file being edited to the beginning, leaving the amended file at the point it has reached.
The first line of the file being edited is again current.
The line that is current at the time the command is given is not copied before the command is obeyed.
This command allows some manipulation of blocks of lines.
Example 1: Assuming line 1 is current, lines 12 - 20 and lines 44 - 63 could be interchanged by the following sequence

```
D 12/43; F64; RETURN; DA 20; F44; RETURN; DA11; F21; DA63;
```

Example 2: If the amended file is to consist of two successive copies of the original, the following sequence will achieve that result

```
BEGIN
FL; M; RETURN
END
```

EXIT Stop editing if an error is found.
The normal procedure of EDIT when a command fails is to attempt to execute the next one. This may lead to an accumulation of errors - particularly in a background job, but also in interactive editing if several commands are entered at once.

The EXIT command sets a switch so that if an error occurs

(1) in background editing, all further editing commands are ignored, and the job finishes as if an END command had been given.

(2) in interactive editing, any further commands that have been read are ignored and the editor waits for another command from the user.
Note. The EXIT command is no longer in force when control is returned to the user, so it should be given again, if wanted.

ERASE Cancel previous amendments.

The first line is again current.

This provides a means of escape if serious errors have been made.

Note. ERASE cannot undo the effect of a RESTART command.

RESET [n] $1 \leq n \leq 3$. If n is omitted it is assumed to be 1.

Reset both files to the position they were in before the last (or second last or third last) time either of them moved.

In effect, this cancels some recent commands.

It cannot cancel an END, ERASE, RESET, RESTART or RETURN command.

The results of this command vary according to circumstances and can best be illustrated by detailed examples.

Example 1: Suppose the first commands of the editing session to be

```
BEGIN
F12; RSE /X/Y/; M2; IB
      Y=Z
£
```

(a) If all the commands have been technically successful, the F12 and M2 commands will have caused movement of both files and the IB command will have moved the amended file.
RESET or RESET 1 will cancel the IB command, virtually deleting the inserted line, but leaving line 14 as the current one.
RESET 2 will cause a return to line 12. The changes made by the RSE command will have been lost, so that IB, M2 and RSE are, in fact, cancelled.
RESET 3 will cause a return to line 1, cancelling all the commands. In this instance, it is equivalent to ERASE.

(b) If the file comprises only 12 lines, the M2 command will fail, without causing any file movement, the IB command will be obeyed and the line inserted before line 12.
RESET or RESET 1 will again cancel the IB command. Line 12 will still be current with the changes made by the RSE command still surviving.
RESET 2 will cause a return to the first line of the file, while RESET 3 will have no result - except an error message.

(c) If the file comprises 13 lines, the M2 command will be obeyed and the end of file reached, so that the IB command cannot be executed.
RESET or RESET 1 will return to line 12, with its amendments lost, RESET 2 to line 1 and RESET 3 will not be obeyed.

Example 2: FB *THE START*; P; ERASE; F8; DS KTRYK ; P;

Assuming the file has at least 8 lines, RESET [1]

will return to the position before execution of F8

(i.e. the first line). RESET 2 or RESET 3 will

produce an error message, since such a command would

be trying to cancel ERASE.

Example 3: D 16/20; M3; RSE AULLAULA ; FC /KER/; P
RESET ; P
RESET

The second RESET will not be obeyed.

Example 4: M3; P
RESET; DS/KER/; M3; P
RESET

If both M commands have been obeyed, both RESET commands are valid.

PRINT commands

These commands cause lines to be printed for verification at the on-line terminal, or, in background editing, on the line-printer.

P [+] Print the current line [and following 10 lines], together with the number of the current line and the total number of lines in the file.

PRINT Print all lines as they are being written to the amended file.

PNEW Print all amended or inserted lines.

PCOMM Print each command as it is executed.

XPRINT Cancel the previous PRINT command.

XPNEW Cancel the previous PNEW command.

XPCOMM Cancel the previous PCOMM command.

OPTION commands

The following commands should be used only when the default options of EDIT (given in brackets after each command) are unsuitable. Where two numbers are given under RAX, the second is that used for background editing. The notes below should be read before parameters are chosen.

		RAX	44MFT
LINE n	Make n (≤132) the number of characters in each line of text.	(80)	(80)
UI n	Read input file for editing from unit n.	(5) (3)	(2)
UO n	Write edited output on unit n.	(10) (2)	(3)
UC n	Read commands from unit n.	(9) (5)	(5)
UA n	Read added lines (in replacement and insertion commands) from unit n. (Note. This does not affect the reading of added lines within a LOOP sequence, which is always from the unit for commands.)	(9) (5)	(5)

In the following commands, "char" is any single character other than a blank or a semi-colon.

TA char	Make "char" the terminator for added lines. This command is needed if an added line has "£" in column 1.	£	£
TL char	Make "char" the terminator for a LOOP sequence This command is needed if the sequence contains a line for insertion with "*" in column 1.	*	*
TI char	Make "char" the terminator of the input file for editing. This command is needed if any line of the file to be edited has "£" in column 1. (See Note 2 below.)	£	£

Notes.

(1) At the start of an editing session, the BEGIN command line is read first, then the whole of the file to be edited and then the second command line.
Any commands that affect the reading of the text file (LINE, UI, TI) or of the second command line (UC) must be given following BEGIN on the same line.

For example: BEGIN; UI 2; LINE 100; UC 5

(2) If commands or added lines are to be read from the same device as the file to be edited, a terminating character for the text file (default option £), in column 1 on a line by itself, must be used as a separator between it and what follows.

(3) EDIT uses unit 1 for the working copy of the file being edited. The working copy of the amended file is on unit 2 (RAX) or unit 8 (44MFT).

Warning.

EDIT always attempts to execute an OPTION command. The user is responsible for the results of changing line-length in midstream, overwriting the original file or reading from the card-punch.

The Standard Use of EDIT under RAX

On-line

```
/ID name
/INPUT
/JOB GO, resour ce information
/FILE DISK=(1,name1),VOL=SYSFIL        } i.e. Work files for EDIT
/FILE DISK=(2,name2),VOL=SYSFIL        }
/INCLUDE BZEDA
/INCLUDE BZEDB
/INCLUDE BZEDC
/INCLUDE file to be edited
/END RUN
(Then in response to ENTER DATA or >)
     BEGIN
       .
       .
Other editing commands
       .
       .
  END
(The edited file should then be saved by)
/SAVE name(lock),SV
```

Background

```
The card deck is:
idpassword
/ID name
/JOB GO,resource information
/FILE DISK=(3,name1),VOL=SYSFIL,DISP=(OLD,KEEP)    (i.e. File to be edited)
/FILE DISK=(2,name2),VOL=SYSFIL,DISP=(NEW,KEEP)    (i.e. Edited file)
/FILE DISK=(1,name3),VOL=SYSFIL                    (i.e. Work file)
/INCLUDE BZED
/INCLUDE BZEDB
/INCLUDE BZEDC
    BEGIN
      .
      .
Other editing commands
      .
      .
  END
/END RUN
```

In both on-line and background jobs, if the file being edited is too large to be accommodated on one cylinder (i.e. 60 512-byte physical records), the NREC parameter must be specified for new files and for work files; if the record size is not 80 bytes, the RSIZ parameter must be given (see RAX manual and Example 1 following) as well as the editor LINE command.

The Use of EDIT Options under RAX

Example 1:

To edit on-line a file of about 700 records (record size 74 bytes) on SYSFIL and to write the edited copy to another SYSFIL file:

```
/FILE DISK=(3,name),VOL=SYSFIL,DISP=(OLD,KEEP)
/FILE DISK=(2,NREC=750,name),RSIZ=74,VOL=SYSFIL,DISP=(NEW,KEEP)
/FILE DISK=(1,NREC=750,name),RSIZ=74,VOL=SYSFIL
/INCLUDE BZEDA
/INCLUDE BZEDB
/INCLUDE BZEDC
/END RUN
(Then in response to ENTER DATA or >)
BEGIN; UI 3; UO 2;  LINE 74
  .
  .
Other editing commands
  .
  .
END
```

Example 2:

To edit in a background job a SYSLIB file and to produce the edited file on punched cards:

```
/FILE DISK=(1,name),VOL=SYSFIL
/FILE DISK=(2,name),VOL=SYSFIL
/INCLUDE BZED
/INCLUDE BZEDB
/INCLUDE BZEDC
   BEGIN; UI5; UO 7
/INCLUDE file to be edited
£                                   (i.e. end of input file separator)
     .
     .
  Other editing commands
     .
     .
   END
/END RUN
```

The Standard Use of EDIT under 44MFT

The card deck is:

```
idpassword
//jobname     JOB        ,limit-information       name,dept.
//SYSOO2      ACCESS      dsname1                 (i.e. File to be edited)
//SYSOO3      ACCESS      dsname2                 (i.e. Edited file)
//            EXEC    XXAMEND
      BEGIN
        .
        .
Other editing commands
        .
        .
      END
/*
/&
```

In this case, dsname1 and dsname2 are assumed to be catalogued sequential disk or tape datasets. If they are not catalogued, the volume specification must be given in the ACCESS statement. If dsname2 is a disk dataset not previously created, ALLOC and LABEL statements must be given in place of the ACCESS statement. (See the 44MFT Background Jobstream Manual and Example 2 following.)

If the file to be edited has more than 1000 records, temporary work files for the editor must be created by the following additional JCL statements:

```
//SYSOOO      ALLOC    XXTA,TAPE=FRESH
//SYSOOl      ALLOC    XXTB,TAPE='lllllll'
```

The Use of EDIT Options under 44MFT

Example 1

To edit a catalogued dataset of about 1200 records, each of 72 bytes and to output the edited file on cards:

```
idpassword
//jobname     JOB          ,limit-information     name,dept
//SYSOOO      ALLOC  XXTA,TAPE=FRESH
//SYSOOl      ALLOC  XXTB,TAPE='lllllll'
//SYSOO2      ACCESS  dsname
//            EXEC  XXAMEND
    BEGIN;    LINE 72;   UO 7
              .
              .
Other editing commands
              .
              .
      END
/*
/&
```

Example 2

To edit a card deck of about 500 cards (80 bytes each) and to write the edited file to a new disk dataset:

```
idpassword
//jobname     JOB      ,limit-information     name,dept
//SYSOO3      ALLOC       dsname,DISK='volid',550,CATLG
//            LABEL       80,76001
/*
//            EXEC  XXAMEND
     BEGIN;UI5
            .
            .
Card deck of file to be edited
            .
            .
£                                    (i.e.  end of input file separator)
            .
            .
Other editing commands
            .
            .
  END
/*
/&
```

Example of complete EDIT job

A simple EDIT job is illustrated. JCL statements for RAX or 44MFT would be added as shown on preceding pages.

File to be edited

```
      SUBROUTINE PUSH(M,N,IRET)
      COMMUN K,MVEC(1OO),NVEC1OO),IRTVEC(1OO)
      K=K+1
      IF(K.LE.1O)GO TO 1O
      WRITE(6,5)
      STOP
   1O MVEC(K)=M
      NVEC(K)=N
      IRRTVEC(K)=IRET
      RETURN
     END
```

Changes wanted

"K" to be changed to "IPT" throughout the file.
In line 2, "U" to be changed to "O" and a round bracket inserted.
In line 4, the first "1O" to be corrected to "1OO".
A format statement to be inserted after the WRITE statement
The first "R" on the third last line to be deleted.
A blank to be inserted before "END".

Commands given

(Some PRINT commands are included to check changes and positions.)

```
  BEGIN
PNEW;  LOOP
FC,K,;   RSE/K/IPT/
*
RESTART;P
F2; P; RS/U/O/;  RS*NVEC*NVEC(*;  M
   FC?1O?;  P;    ISA/C/O/;
    FB/S/; P;   IB
    5 FORMAT(2OH ***STACK IS FULL***)
£
   FC/R/;  P;    DS?R?;
FL;  P  ;   ISB/E/ /;   END
```

Edited file

```
      SUBROUTINE PUSH(M,N,IRET)
      COMMON IPT,MVEC(1OO),NVEC(1OO),IRTVEC(1OO)
      IPT=IPT+1
      IF(IPT.LE.1OO)GO TO 1O
      WRITE(6,5)
   5  FORMAT(2OH ***STACK IS FULL***)
      STOP
   1O MVEC(IPT)=M
      NVEC(IPT)=N
      IRTVEC(IPT)=IRET
      RETURN
      END
```

Messages during execution

```
NNN          COMMUN IPT,MVEC(100),NVEC100),IRTVEC(100)
NNN          IPT=IPT+1
NNN          IF(IPT.LE.10)GO TO 10
NNN       10 MVEC(IPT)=M
NNN          NVEC(IPT)=N
NNN          IRRTVEC(IPT)=IRET

XXX    PARAMETER OF FOLLOWING COMMAND NOT FOUND FROM LINE NO.  9 TO
XXX    END OF FILE:
XXX    FC,K,
XXX    LAST LINE NOW CURRENT.  INTERVENING LINES COPIED TO NEW FILE.

       SUBROUTINE PUSH(M,N,IRET)
XXX    CURRENT LINE IS NO.   1. NO. OF LINES IN FILE IS   11.

       COMMUN IPT,MVEC(100),NVEC100),IRTVEC(100)
XXX    CURRENT LINE IS NO.   2. NO. OF LINES IN FILE IS   11.

NNN          COMMON IPT,MVEC(100),NVEC100),IRTVEC(100)
NNN          COMMON IPT,MVEC(100),NVEC(100),IRTVEC(100)

       IF(IPT.LE.10)GO TO 10
XXX    CURRENT LINE IS NO.     4.  NO. OF LINES IN FILE IS 11.

NN           IF(IPT.LE.100)GO TO 10

       STOP
XXX    CURRENT LINE IS NO.  6.  NO. OF LINES IN FILE IS   11.

NNN        5 FORMAT(20H ***STACK IS FULL***)

       IRRTVEC(IPT)=IRET
XXX    CURRENT LINE IS NO.   9.  NO. OF LINES IN FILE IS   11.

NNN          IRTVEC(IPT)=IRET

       END
XXX    CURRENT LINE IS NO.   11.  NO. OF LINES IN FILE IS  11.

NNN          END

XXX    LENGTH OF AMENDED FILE IS   12 LINES.
```

APPENDIX B

E D I T

A Program for Amending Text Files

Brief Guide

INTRODUCTION

This brief guide to EDIT deals only with its interactive use under RAX.

For background use under RAX or 44MFT, see the full User's Guide. This also gives details of additional commands and available options.

File to be Edited.

This should have been saved previously by a /SAVE command.

Editing is carried out on a copy of the file, so the original is unchanged.

Each line is assumed to be 80 characters in length.

Line numbers may be verified before running the editor by /DISPLAY name.

(Warning. For technical reasons, the editor tests for a pound sign (£) in column 1 of each line of the file. If there is any possibility that this occurs in your file, see the TI command in the full User's Guide.)

Edited File.

This should be saved at the end of the editing session by /SAVE name(lock),SV.

Starting an Editing Session:-

```
/ID name
/INPUT
/JOB GO,resource information
/FILE DISK=(1,name1),VOL=SYSFIL
/FILE DISK=(2,name2),VOL=SYSFIL
/INCLUDE BZEDA
/INCLUDE BZEDB
/INCLUDE BZEDC
/INCLUDE file to be edited
/END RUN
```

(The editor will then print a prompt for the command "BEGIN".)

SUMMARY OF COMMANDS

BEGIN	Begin editing session. (This must be given.)
END	End editing session. (This must be given.)
F n	Find line no. n.
FB /string/	Find line beginning with "string".
FC /string/	Find line containing "string".
M[n]	Move 1 line [n lines].
D[+n]	Delete current line [and following n lines].
DB /string/	Delete from current line to before line beginning with "string".
DC /string/	Delete from current line to before line containing "string".
DS /string/	Delete first occurrence of "string" in current line.
IA	Insert line(s) following IA after current line.
IB	Insert line(s) following IB before current line.
ISA /string1/string2/	Insert "string2" after first occurrence of "string1" in current line.
ISB /string1/string2/	Insert "string2" before first occurrence of "string1" in current line.
EXIT	Stop editing if an error is found.
ERASE	Cancel all previous amendments and make the first line current again.
P[+]	Display current line [and following 10 lines].

GENERAL NOTES

Current Line

In editing, a single pass is made through the file.

The current line is the one being "pointed at" or "looked at" at any given time.

Commands may refer to the current line or may move the file forward so that a later line is current, but may never move it back to an earlier point.

At the start of an editing session, the first line of the file is current.

Typing of Commands

Spacing is not important - except within character string parameters.

Commands may be entered (in response to ENTER DATA on the 2260 or > on the teletype) either one at a time or several to a line, using the semi-colon as a separator.

e.g. either

```
F 12
P+
DS /AND/
```

or

```
F12; P + ; DS/AND/
```

(Exception. BEGIN must be typed on a line by itself.)

(Warning. There is the danger of an accumulation of errors if several commands are entered and executed together - but see EXIT and ERASE commands.)

Text for insertion following IA and IB commands must start on the next line and not continue on the same one.

Character String Parameters

These are referred to as "string", "string1" and "string2" in the description of commands.

They may be of any length up to 80 characters, but should be as short as possible.

The delimiters of a string - represented by the solidus (/) in the definitions of the commands - may be any character (except blank or S or E) not contained in the particular string.

All blanks in character strings are significant except leading blanks in FB and DB commands.

Integer Parameters

These are referred to as "n" in the description of commands. They must be positive and unsigned.

DESCRIPTION OF COMMANDS

General commands

BEGIN — Begin the editing session.

This command must be given. It must be typed on a line by itself.

END — End the editing session.

This command must be given.

If the end of file has not been reached when this command is entered, the remainder of the file is copied at this point.

P[+] — Display the current line [and the following ten lines], together with its number and the total number of lines in the file, for verification.

EXIT — Stop editing if an error is found and wait for the next command from the user.

The normal procedure of the editor when a command fails is to attempt to execute the next one.

This command avoids an accumulation of errors when several commands have been entered at once. If one of these commands fails, the remaining commands are ignored and the program returns control to the user.

The EXIT command is no longer in force once control is returned, so EXIT should be retyped if wanted.

ERASE — Cancel all previous amendments and make the first line current again.

This provides a means of escape if serious mistakes have been made.

FIND and MOVE commands

These copy up to, but not including, the specified line, which becomes the current one.

F n — Find line no. n.

For example:
```
F24
F 229;
```

FB /string/ — Find line beginning with "string". Leading blanks in the line and in "string" are ignored.

For example: The Fortran statement

```
20 II=II+1
```

would be identified by either of the following

```
FB  *20*
FB &        20&
```

but not by

```
FB ?20    ?
```

FC /string/ — Find line containing "string".

All blanks are significant.

For example: The line

```
IF(X.EQ.Y)GO TO 65
```

would be identified by either of the following

```
FC /GO TO/
FC *GO TO *
```

but not by either of the following

```
FC / GO TO/
FC *GOTO*
```

Note. FB is more efficient than FC and should be used when possible.

M[n] Move n lines beyond current one.

If n is omitted, it is assumed to be 1.

For example:
```
M
M13;
```

DELETE commands

D[+n] Delete current line [and following n lines].

The line after the last deleted one become current.

For example:
```
D;
D+4
```

DB /string/ Delete from current line to before line beginning with "string" - which becomes current.

See notes under FB.

DC /string/ Delete from current line to before line containing "string" - which becomes current.

See notes under FC.

DS /string/ Delete first occurrence of "string" in current line. The same line remains current unless all non-blank characters have been removed, in which case the line is considered deleted and the next line becomes current.

For example: Following the command

```
DS / VERY/
```

the line

```
IT IS VERY STRANGE AND VERY TRUE.
```

becomes

```
IT IS STRANGE AND VERY TRUE.
```

If the command is repeated, the line becomes

```
IT IS STRANGE AND TRUE.
```

INSERT commands

Insertion of complete lines

IA — Insert line(s) following IA after the current line.

The next line of the file becomes current.

IB — Insert line(s) following IB before the current line.

The current line remains the same.

The first line for insertion must not be typed on the same line as the command.

The end of the insertion must be indicated by a pound sign (£) in column 1 on a line by itself.

(Note. If any of the inserted lines has a pound sign in column 1, see the TA command in the full User's Guide.)

```
Example 1:        F12; IA
                    First inserted line
                        . . .
                        . . .
                    Last inserted line
                  £
```

Several lines are inserted after line 12 of the file.

Line 13 is the new current line.

Example 2: If IA in the above example were replaced by IB, the lines would be inserted before line 12, which would remain current.

```
Example 3:        F26; D +4;  IB
                     Replacement line(s)
                  £
```

Lines 26 to 30 of the file are replaced with another line, or other lines. Line no. 31 becomes current.

INSERT commands (cont.)

Insertion within a line

Insertion of characters within a line causes the loss of space characters at the end of the line. If any non-blank characters overflow, a new line is created to hold them and it is considered the new current one. Normally the same line remains current.

ISA /string1/string2/ Insert "string2" after first occurrence of "string1" in current line.

ISB /string1/string2" Insert "string2" before first occurrence of "string1" in current line.

Example 1: Following the command

ISA *VERY*, VERY*

the line

IT IS VERY STRANGE AND VERY TRUE.

becomes

IT IS VERY, VERY STRANGE AND VERY TRUE.

If the command is repeated, the line becomes

IT IS VERY, VERY, VERY STRANGE AND VERY TRUE.

Example 2: To replace the first "HAPPY" in the following sentence by "DELIGHTED"

HE WAS HAPPY THAT THEY WERE HAPPY.

it would be possible to type

DS /HAPPY/; ISB / T/DELIGHTED/

or

DS /HAPPY/; ISA /S /DELIGHTED/

To replace the second "HAPPY" in the original sentence, either of the following lines would serve.

DS /HAPPY./; ISA/RE /DELIGHTED./

DS /E HAPPY/; ISB/./E DELIGHTED/

Example of a Complete EDIT Job

File to be edited

```
      DIMENION MAIN(880)
    5 READ(5,10,END=15)MAIN
   10 FORMAT(80A1)
      WRITE(2,10)MAIN
   15 ENDFLE 2
      REWIND 2
      REWIND 2
   18 READ(2,10,END=25)MAINN
      WRITE(6,20)MAIN
     GO TO 18
   20 FORMAT(1X,80A1)
   25 STOP
```

Changes wanted

In first line, "S" to be inserted and "8" deleted.
Statement to be inserted between lines 4 and 5.
"I" to be inserted in line 5.
One of the "REWIND 2" lines to be deleted.
In line 8, "N" to be deleted.
In line 10, a blank to be inserted before "GO".
An "END" statement to be inserted after last line of file.

Commands given

```
   BEGIN
 ISA/N/S/;  DS/8/;P
FB/15/;P;   IB
       GO TO 5
£
ISB?L?I? ;P; M; D
  FC/NN/; DS/NN/; ISA/AI/N/;P
M2; ISB/G/ /;P
 FL; IA
       END
£
END
```

(The P commands allow changes and positions to be checked.)

Edited File

```
      DIMENSION MAIN(80)
    5 READ(5,10,END=15)MAIN
   10 FORMAT(80A1)
      WRITE(2,10)MAIN
      GO TO 5
   15 ENDFILE 2
      REWIND 2
   18 READ(2,10,END=25)MAIN
      WRITE(6,20)MAIN
      GO TO 18
   20 FORMAT(1X,80A1)
   25 STOP
      END
```