Behavior Research Methods & Instrumentation
1974, Vol. 6, No. 2, 159-166

SESSION VI
TUTORIAL

JOSEPH B. SIDOWSKI, *University of South Florida*, **Presider**

On-line computer text processing: A tutorial*

RICHARD C. ROISTACHER
Center for Advanced Computation and College of Commerce
University of Illinois at Urbana-Champaign, Urbana, Illinois 61801

The use of an on-line computer system for preparing manuscripts affords the scholar many advantages. This tutorial discusses text editing, file and output processing systems, and how they may be used for producing and disseminating scientific documents.

Text processing is a powerful computing utility whose user community is rapidly growing. Text processing as used here refers to the storage and editing of manuscripts maintained as computer files of text and the use of computer programs to format those manuscript files into documents. Text processing as used in this tutorial does not include either information retrieval, in which a file of information is scanned for a desired entry, or such areas as computer-based analysis of literary text.

The aim of this tutorial is to introduce computer text processing to a reader who has had some experience with on-line computer systems, but who is not necessarily a programmer. While this paper reviews several systems and machines, it is not meant to be a complete review of the field. The particular machines, systems, and criticisms presented here are drawn primarily from the author's experience, rather than from an exhaustive literature search.

Text processing services are what might be called external computing utilities, in that like BASIC or teaching programs, they provide sophisticated computing services to clients who are not necessarily programmers or even members of the usual computer user community. Text processing systems have been in the libraries of several operating systems for many years. Until recently, text processing has not been widely used because the programs required considerable computing sophistication to use, and because they were quite expensive to run. In academia, it was possible for interested computing center "insiders" to use text processing systems because they had access to the necessary resources, and because there were not so many text users among the insiders as to overload the available resources. However, the potential market for computer text processing was so wide as to make university computing administrators fear for their ability to support a drastically increased community of users, most of whom would have to operate on internal funds.

Text processing systems have now evolved to the point where they should find more general use, especially among those who use on-line systems in the course of their other work. Text processing systems can be found on many time-sharing systems, including IBM's Time Sharing Option for OS/360, Digital Equipment Corporation's TOPS, and Bolt, Beranek, and Newman's TENEX for the PDP-10, the Burroughs B-6700 MCP, the University of Michigan's Michigan Terminal System, and the Massachusetts Institute of Technology's Multics.

TEXT PROCESSING SYSTEM COMPONENTS AND CHARACTERISTICS

A computer text processing system has three major components: a file system, an editor, and an output processor. The file system is the computer resource for cataloging and storing information which is not in the machine's memory. Information may be stored on a variety of devices, including magnetic disks, drums, and tapes. Most text files are stored on a rapid access device such as a disk while they are being edited and processed, and are then archived onto magnetic tape.

An editor is a program for the insertion, scanning, and deletion of text in computer files. Most editors are used primarily for editing programs maintained as disk files, rather than as decks of punched cards. Editors have evolved from primitive programs, which allowed only the replacement of specified lines in a file into programming languages in their own right, capable of

*This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the U.S. Army Research Office-Durham under Contract No. DAHC04-72-C-0001, and in part by the College of Commerce and Business Administration, University of Illinois at Urbana-Champaign.

complex sequences of operations on text files.

Once text has been input to the file system and has been corrected with an editor, it must be arranged into a proper output format through the use of an output processor, a program for formatting, justifying, and printing text. Output processors are extremely complicated programs, approaching compilers in complexity. An output processor uses commands inserted in the stream of text to control the formatting of the finished document.

Text processing offers the scholar a number of powerful advantages in the preparation and dissemination of scientific papers. The most obvious advantage is that error correction tends to be cumulative. Once corrections are made, they stay in the file, and thus typographic errors are not reintroduced into corrected copy, as is the case when drafts must be retyped by hand. It is also possible to find and correct all errors of the same kind in a single editing operation. The ability to insert, delete, and rearrange text without regard to page boundaries and numbering allows the scholar to revise his paper far more easily than is the case when he must physically cut, paste, and retype his manuscript.

Bibliographic work is especially facilitated by the computer, even when no formal information retrieval or indexing procedures are used. It is possible to add to a file of bibliographic references as each new reference is found, making sure that each new entry in the file is correct. Once a paper has been completed, it may be scanned for all bibliographic citations, e.g., by printing all lines containing the character string "(19", or ",19", and the relevant references extracted from the bibliographic file.

One advantage of computer text processing, which becomes apparent only after some experience, is the freedom from paper copy itself. There is no longer any "original" which must be preserved from harm. Duplication of an original becomes a relatively simple task, as a tape copy of a several hundred page document stored on magnetic tape can be made in a few minutes, and far more cheaply than such a document could be Xeroxed.

Dissemination of a document via the computer may not be important when all of the users of a time-sharing system are in the same institution. However, when users are spread over a wide area, as in the case of a regional or national computer network, the system itself provides the fastest possible way of transmitting a manuscript to an interested reader.

Computer text processing, however, cannot be said to be an unmixed blessing. The obvious disadvantage is the additional cost to the user, who must invest in a terminal and extra computer time. The faithful Model 33 Teletype, while technically usable as an input to a text processing system, is too slow and fatiguing to fill the bill and must be replaced by a more expensive device. Another disadvantage is that on-line computer systems are usually rather susceptible to Murphy's law that "If anything can go wrong, it will" and will tend to be least available when they are most needed. Since text processing tends to be addictive, an unreliable system can be a major handicap. A third disadvantage arises directly from one of the advantages of text processing, the ability to make instant backup copies of document files. Unless the user exercises a good deal of self-discipline, it is possible to become lost in a welter of different versions and copies of a document, leading to confusion, delay, expense, and sometimes loss of a document.

FILE SYSTEMS

File Operations and Capabilities

The power and flexibility of a computer's file system is of great importance to the text processing user, who will often want to do rather complex copying and concatenation operations when such capabilities are available. It is often best practice to keep a long document in several relatively short files, which are concatenated (attached end to end) at the time they are fed to the output processor. Sometimes it is possible to concatenate a file to the interior of another at run time, a useful feature for inserting tables and diagrams.

Numbered-Line Files

Direct access files can be loosely classified into numbered-line and stream formats. A numbered-line file, as its name indicates, has a number assigned to each of its lines. It is possible to insert and delete lines from such a file, but each line is a separate entity. A numbered-line file need not have its numbers displayed when it is printed, but it is still stored internally as separate, numbered lines. In a numbered-line file, the line numbers are stored in a directory and do not change when a line is inserted or deleted. Instead they are explicitly reordered at the user's command.

Stream Files

A stream file is stored as a continuous stream of characters. Lines of the file are delimited by a carriage return and linefeed character sequence or by a special end-of-line character defined by the operating system. The major advantage of such a file is that it is extremely easy to edit across lines, since all that is required is the insertion or deletion of a line delimiting character. The major disadvantage of stream organization is that there is no fixed way to refer to parts of the file. It is possible for the system to count the number of line delimiters between the head of the file and a given point, but any change in the number of such delimiters will change the "line number" of a given piece of text.

EDITORS

Basic Characteristics

The characteristics of an editor must match those of the file system on which it operates. It is possible to classify editors as being either line or character oriented, and as either line number editors or context editors.

Line Oriented Editors

Line oriented editors treat lines as separate units and will not combine lines of text. They are far more common than character oriented editors, which in most cases treat the line delimiting character as they would any other character. The chief advantage of line oriented editors is their ease of use. The system will keep track of lines for the user, who is free to add, delete, or change lines without having to maintain their boundaries himself.

Line oriented editors can be directed to specific line numbers in a numbered-line file system. Otherwise, it is necessary to address text exclusively by context. In the former case, it would be possible to tell the editor to delete Line 44.5, while in the latter case it would be necessary to tell the editor to delete the line containing the string "To be or not." Examples of line oriented editors are the TSO editor, *ED, residing in the Michigan Terminal System, and Multics QEDX.

Character Oriented Editors

The only character oriented editors in general use are variants of TECO, resident on the PDP-10 operating systems, and in Multics. Character oriented editors are usable only with stream files, since they process line delimiter characters. It is possible to perform certain operations across lines more easily with a character oriented editor, but TECO offers few advantages over a good line editor.

Examples of scanning, insertion, and replacement operations using a line number editor, a line context editor, and a character oriented editor appear in the appendix.

Editor Functions

The basic function of an editor is to construct a text file. In its most basic form, an editor simply reads a line of text from an input device, such as a terminal or card reader, and writes it onto a machine-readable storage medium. In numbered-line file systems, a primitive editor can also be used to add or delete particular lines, addressing them by number. In some computer time-sharing systems, these functions are built into the command language, and the editor program need not be explicitly run in order to make minor changes in a file.

For example, insertion of the text "To be or not to be" between Lines 305 and 306 of a file called FOO would be done in a sequence of (hypothetical) system commands such as

```
*OPEN FOO
305.5 To be or not to be
*CLOSE
```

Beyond its basic use in creating files, an editor's basic function is to find and replace strings of text. Such searches and replacements may be done for a single case, for all occurrences in a given range of line numbers, for a given number of occurrences, or for all occurrences of the string in a file. For example, to find the string "not" in a file, a command of the form

```
*FIND /not/
```

is typed. A modification of the command can be used to locate multiple occurrences of the string.

To replace the word "not" with the word "just" in the text example shown above would be done with a command of the form

```
*REP /not/just/
```

resulting in the string

```
305.5 To be or just to be
```

Finding and changing multiple strings is the capability on which almost all editor functions are built. In some editors, it is possible to use special string expressions, so that replacements are made only when specified conditions are met. It is possible to write programs in some editor command languages which allow for extremely complicated editing operations. For example, it is possible to write a program in TECO which will do most of the work of reformatting an American Psychological Association formatted bibliography into one in American Sociological Association format.

Most sophisticated editors have the capacity for allowing the user to recover from editing steps when he has made a mistake. In most systems, the editor works on its own copy of the file, explicitly writing the edited copy onto the disk or tape at the user's command. If a mistake has been made, it is possible to copy some or all of the original text from the disk into the editor's copy of the file. The MTS editor, which works on the disk copy of the file, keeps a record of changes made and has a RESTORE command, which may be invoked for any line or group of lines to return them to their original state.

Choosing an Editor

The advent of a large text processing user community is putting greater and greater demands on the developers of editors. Editors which seem perfectly satisfactory for correcting programs have proven far from satisfactory for the far more complex task of editing large bodies of free-format text. As a result, we may look forward to the continuing development of editors, some of which may result in something useful.

The prospective text processor who is a member of a larger user community may find himself with a choice of several editors of several types. Since an editor is the single program an on-line text processing user uses most, it is wise to begin by choosing a single editor and sticking with it until its use becomes second nature.

Accordingly, some rough-and-ready criteria for choosing an editor are: (1) Take the claims of programmers (and especially system programmers) with many grains of salt. Document text is much more difficult to edit than is program text, a fact of which few programmers are yet aware.

(2) If possible, choose a line oriented editor with numbered-line files. Text references are far easier when line numbers are available, especially when one is discussing a manuscript over the telephone. If no reasonably convenient and powerful line number editor is available, look for a line oriented context editor, which will work with stream files. Finally, go to a character oriented editor, which may be extremely powerful, but somewhat harder to use.

(3) Choose an editor whose basic find-and-replace features are simple, yet powerful. For example, an editor which requires that the command REPLACE" be written out in full and in capital letters is not nearly as convenient as one which will accept "REPLACE," "REP," "rep," or "r" as equivalents. Similar elegance can be found in other basic aspects of the command language. For instance, suppose that one wished to change the first two instances of the word "not" to the word "just" in a file. In one editor, the only way to do so would be to give the commands

```
r 'not' 'just' 1 999
r 'not' 'just' 1 999
```

Another editor could accomplish the task in two commands with

```
r 1 /not/just/
r
```

The second of the two examples illustrates several design features which are sadly lacking in the first. First, there is no set delimiter character, such as the required "'" of the former example. Instead, the first nonblank character in the string is recognized as the delimiter for that string. Second, only a single delimiter is needed between the "find" string and the "change" string, rather than the three-character delimiter (including two uppercase characters) in the first example. Third, in the latter case, the editor remembers the previous strings in the "replace" command and uses it in all subsequent "replace" commands if no new pair of strings is given. Fourth, the latter editor has a "line pointer," a counter which indicates which line is being edited. In the former example, the user was not sure where the first occurrence of "not" could be found, so he specified that the change would be made in Lines 1-999. Since the editor has no line pointer, the user had to repeat the entire line range specification in the second command. In the second editor, the line pointer is set at Line 1 in the first command, and the file is searched until "not" is found and replaced. The line pointer is now set at the line at which the replacement has occurred, and the next search begins at this line. Many editors which have no line pointer either require a line number as a part of each command or assume that the bottom of the file is the default when no number is given.

Relatively simple command language features, which arise not so much from elegant and difficult programming as from wide and innovative user experience, make the difference between editors which are helpful and those which are a constant trial to the user.

(4) Do not be overly concerned with the operating speed of an editor if its command language is good. Editors use very little of the computer's time in comparison with their demands on the user's time and temper.

OUTPUT PROCESSORS

An output processor is a program for transforming a free-form file of text into a document in standard form. The basic function of the output processor is to fill text lines and pages in the finished document by concatenating and splitting lines of the original text file. The program keeps track of page numbers and formats and has facilities for the automatic printing of headers, footers, and page numbers. Most output processors will also justify output text lines so that the right margins of the finished document are smooth, and have a facility for indicating symbolically whether a letter or word is to be in upper- or lowercase, allowing the use of a Teletype or other uppercase terminal. The TENEX RUNOFF program, for instance, normally converts all uppercase letters to lowercase unless they are preceded by a "^." The RUNOFF user could type "^MY NAME IS ^FRED.", which the program would copy into the document as "My name is Fred." Facilities for handling uppercase-only text are no longer being included in some of the latest output processors, as the Teletype is giving way to more versatile terminals.

Processor Command Languages

The extensive repertory of commands necessary to control an output processor gives this type of program the complexity of a compiler. Commands are generally inserted into the file of text as it is entered into the original file. Commands are generally distinguished from text by a leading character, such as FORMAT's leading right parenthesis. In some output processors, a command must be on a separate line, while in other systems, commands may be interspersed freely between words of the text. Like the command languages of editors, output processor languages come in all degrees of sophistication and complexity. For example, consider the problem of beginning a new page with the following text:

```
                CHAPTER III

  New heading. This is a new chapter . . .
```

The TENEX RUNOFF program on the PDP-10 would produce this output from the following:

```
.PAGE
.TEST PAGE 8
.CENTER
CHAPTER III
.PARAGRAPH
^&New heading.\&
This is a new chapter . . .
```

The IBM FORMAT program would produce the same output from:)SM CHAPTER III)MPU New heading.)U This is a new chapter . . .

The TENEX RUNOFF program requires that commands be on separate lines and spelled out in capitals. The commands to the FORMAT program consist of single letters or of letter-number combinations, and each string beginning with ")" is interpreted as a command string. Thus, ")S" is the command to start a new page, ")P" starts a new paragraph, all text between ")U"s is underlined, and all text between ")M"s is centered.

Some output processors combine their command language with the text itself. The Multics runoff program treats an empty line (one consisting of nothing but a carriage return) as a command to end the present line immediately and begin a new line of output. If the next line is also empty, the same action is taken, but if the next line begins with a horizontal tab character, this tabulation is printed. Thus, a new paragraph is input to the Multics runoff program just as it would normally be typed, as a blank line followed by a tab.

Advanced Output Processor Features

Advanced output processor features include the ability to move text in order to keep it together, to evaluate symbolic references and expressions, and to hyphenate.

Most output processors have a command which forms "keeps," groups of text lines which are to be kept on the same page. Sometimes a block of blank lines is reserved for the insertion of a figure. When a keep command is encountered, the output processor counts the number of lines remaining on the current output page and skips to the next page if insufficient space remains. A far more complex problem is the establishment of "floating keeps" whose position in the text stream is altered by the processor. In a simple keep, a preceding page will simply have blank lines at the bottom if it has been necessary to place the kept text on a new page.

In a floating keep, the text immediately following the kept portion is moved ahead to fill the gap at the end of the preceding page. A footnote is handled in much the same manner as a floating keep. It is inserted next to its reference in the input text, and the output processor moves the footnote to the bottom of whatever page the footnote reference occurs on.

Some advanced output processors include a facility for maintaining counters and for evaluating symbolic references. The program maintains a set of counters which can be incremented on command and whose current values can be substituted into the text on output. Thus, instead of having to refer to "Table 4" at the time text is input, the author can use the text "Table %Tab%" where "Tab" is a symbolic reference to the current value of the table counter. A reference to "Tab" is made in a way which adds 1 to its current value when it is used in the label of a table. Thus, it is not necessary to renumber tables or figures as additions or deletions are made. The same counter is referenced but not incremented when it is used in text which refers to the previous table.

Symbolic reference facilities are not included in most output processors. The current MTS FORMAT program includes a set of footnote and equation counters. The Multics runoff program has facilities for the evaluation of extremely complex symbolic expressions, which include the substitution of character strings as well as numbers.

No generally satisfactory method for hyphenating and breaking words has yet been devised. Most output processors fill a line and then justify it without breaking words, sometimes producing an occasional line containing an unbearable amount of white space. The Multics runoff program has a facility for allowing the user to introduce his own hyphenating routine which, in most cases, scans a list of words and their hyphenation points. Such a repeated search can become extremely expensive, and is thus not a part of most simple output processors.

IBM's ATS/360 and its variants approach the hyphenation problem rather successfully from the other side. An input text line which ends with a hyphen is considered to have its last word continued onto the beginning of the next line. If the split word fits completely into an output line, the hyphen is deleted and the word is reassembled. Otherwise, it is used in its hyphenated form, split between lines. Hyphens used in other ways are not placed at the end of input lines.

Choosing an Output Processor

The interested user will usually have the opportunity to choose from a number of output processing programs. Some considerations in selecting an output processor are: (1) Many different processors have the same name. There are at least three versions of IBM's FORMAT, with radically different features and performance. There are at least three programs named RUNOFF, written in at least two different languages. Look at the program's documentation and pedigree to make sure you know what you are dealing with. (2) A wide search for a good processor is often worthwhile. Unlike editors, which are heavily machine- and system-dependent, it is often possible to move output processors between machines relatively easily. (3) Pay attention to the performance of the processor, as well as the power and convenience of its facilities. Output processors can become extremely expensive to operate. (4) Unless you are reviewing this paper as a referee, do *not* attempt to write your own output processor. You have neither the time nor the money to do the job decently.

Two IBM Systems

Two systems produced by IBM and fairly widely distributed should be mentioned. The Administrative Terminal System, ATS/360, is a small document processing system designed to be used by secretaries. ATS will produce reasonable documents, but at a fairly heavy cost. In order to make the system immediately comprehensible to their conception of a machine-hating (and partially retarded) secretary, the designers of ATS made some decisions which go heavily against the grain of good computing system design. ATS has no editing capabilities beyond the replacing of strings in a specified line. It is not possible to tell ATS to find a particular character string, but only to go to a line whose number is known and perform the replacement. Even though ATS uses a line number editing procedure, its line numbers change as lines are added and deleted, forcing the user to edit from back to front. Finally, ATS has no separation between its file system, editor, and output processor, which are all mixed together in a hopeless mass. ATS/360 is better than nothing, but almost anything else is better than ATS. Some commercial time-sharing services offer augmented versions of ATS which have more powerful editing facilities and output capabilities, and these make the system more easily usable.

Another IBM product is a series of programs known collectively as TEXT/360. This set of PL/1 programs is used to produce the IBM documentation and other book-length pieces. They are not for the short (under 50 pages) piece of text nor for the inexperienced user.

The bibliography lists some documentation for a variety of text processing facilities together with some comments. However, the reader will have to be his own judge of whatever he finds available.

EDITING OF OUTPUT TEXT

In most cases, it is unwise to run the output of a processor directly onto a terminal or printer. Many times, the program will produce some embarrassments which may be corrected with a little editing. It is almost always preferable to direct the output of a processor into a file, print the file, and make corrections as necessary. Failure to edit output text will result in greatly increased expense, as editors are considerably less expensive to run than are output processors. It is important that changes made to the output text be added to the input version, in case the document must be reprocessed later.

Sometimes output text editing is necessary because the processor lacks the necessary capabilities. For example, the TENEX RUNOFF program has the irritating habit of centering all centered lines around Space 30, even when the margins have been set so that Space 30 is not the center of the page. The author corrects this on input by beginning each line which is to be centered with a special character such as "@." When the output text is edited, all "@"s are replaced with a number of spaces sufficient to shift the centered line to the proper center of the page. Other such editing tricks are dependent on the foibles of each system and output processor.

FUTURE PROSPECTS AND FINAL WORDS

Computer text processing is not yet trivial, but the increasing reliability and sophistication of computer systems has made text processing feasible for an increasing number of computer users. The advantages of a well-behaved text processing system far outweigh the system's disadvantages. Once a system is well established, it is possible to delegate the mechanics of text entry and editing to a secretary.

As computer networks gain more and more clients, one's library of manuscripts will be instantly accessible to a nation- and worldwide set of colleagues for consultation and exchange of views. This paper was written on a portable terminal in the author's study at home in Urbana. It was stored and processed on a PDP-10 at the Information Sciences Institute at the University of Southern California, where it is presently resident on magnetic tape. The final copy was printed on a Model 37 Teletype located in the machine room of the Center for Advanced Computation at the University of Illinois. The arrangement of author, computer, and printing facility was made possible by the ARPAnet, a nationwide computer network, which links the University of Illinois with the University of Southern California. The author's situation, while presently somewhat avant garde, will soon be close to the norm for the scientific community. Computer text processing by itself is a great convenience. Combined with computer networking, it is part of a revolution of knowledge.

APPENDIX

This appendix consists of three versions of the same editing and formatting job. The first version is edited using the PLORTS system on the University of Illinois's System/360-75 and formatted on the Illinois version of IBM's FORMAT. The second version is edited in TENEX TECO and formatted using RUNOFF on the PDP-10 at the University of Southern California's Information Sciences Institute. The third version is edited using the

Multics version of QEDX and formatted using Multics runoff.

The TENEX and Multics jobs are shown entered and printed on a Model 37 Teletype. The Illinois job is shown entered on a Model 33 Teletype and run off on a line printer using a TN (upper- and lowercase) print train.

Each editing job consists of the same sequence of operations: (1) The editor is invoked and the file is read into it. (2) The file is scanned to see if the word "page" has been mistyped. A misspelling is found. (3) All instances of the misspelling are corrected and printed as the corrections are made. (4) The entire file is printed, and a wrong line is found. (5) The offending line is deleted. (6) The line is replaced. (7) The text immediately containing the correction is printed in order to verify the correction. (8) The file is written (or closed, as appropriate) and the editor terminated.

Each example consists of a page of editing, followed by a page showing the results of the output processor. No postprocessing edit has been done in order to show some differences in the output processors and to keep the examples clearer.

OS/360 PLORTS Editing Example

```
OPEN EX3    ←——————————  1
S PAEG      ←——————————  2
   7.000 RIGHT PAEG NUMBER
R PAEG PAGE 1 99 ←—————  3
   7.000 RIGHT PAGE NUMBER
   9.000 LINES PER PAGE 30
  22.000 BE KEPT ON THE SAME PAGE.
L    ←—————————————————  4
   1.000 /*ID FORMS=1100
   2.000 //TRIAL EXEC FORMAT
   5.000 //SYSIN DD *
   6.000 WIDTH 60
   7.000 RIGHT PAGE NUMBER
   8.000 CAPS AUTOMATIC
   9.000 LINES PER PAGE 30
  10.000 REPEAT TITLE
  11.000 SENTENCES 2 BLANKS
  12.000 START TEXT 5 5
  13.000 FOOTER
  14.000 (1) [THIS FOOTNOTE REFERS TO THE DIAGRAM.
  15.000 )E
  16.000 GO
  17.000 )MF A TEXT PROCESSING EXAMPLE )MLP
  18.000 THIS IS AN EXAMPLE OF TEXT PROCESSING WHICH SHOWS HOW TEXT
  19.000 CAN BE INSERTED IN FREE FORM AND THEN FORMATTED BY THE
  20.000 COMPUTER.
  21.000 )P FOLLOWING THIS PARAGRAPH IS A BLOCK DIAGRAM (1) WHICH MUST
  22.000 BE KEPT ON THE SAME PAGE.
  23.000 IF THERE IS NO ROOM FOR IT ON  PAGE 1, THEN IT ON WILL BE
  24.000 PRINTED ON PAGE 2.
  25.000 )LLLW7A
  26.000          !24--------!25     !24--------!25     !24--------!25
  27.000          !33        !33     !33        !33     !33        !33
  28.000          !33        !33     !33        !33     !33        !33
  29.000          !33  [BEGIN !33---->!33 [MIDDLE !33---->!33   [END  !33
  30.000          !33        !33     !33        !33     !33        !33
  31.000          !33        !33     !33        !33     !33        !33
  32.000          !22--------!23     !22--------!23     !22--------!23
  33.000 )
  34.000 ************************** NOSTALGIA ISN'T WHAT IT USED TO BE.**
***
  35.000 TITLE
  36.000 )F TEXT PROCESSING EXAMPLE )FE
  37.000 GO
  38.000 )E
  39.000 /*
34 )LM [DIAGRAM 1. )MV ←——  5,6
L 33 35   ←————————————————  7
  33.000 )
  34.000 )LM [DIAGRAM 1. )MV
  35.000 TITLE
C ←———— 8
```

OS/360 FORMAT Output Example

```
                   A Text Processing Example



     This is an example of text processing which  shows  how
text  can be inserted in free form and then formatted by the
computer.

     Following this paragraph is a block diagram  (1)  which
must  be  kept on the same pago.  If there is no room for it
on page 1, then it ou will be printed on page 2.


          r---------,     r---------,     r---------,
          |         |     |         |     |         |
          |         |     |         |     |         |
          |  Begin  |---->| Middle  |---->|   End   |
          |         |     |         |     |         |
          |         |     |         |     |         |
          L---------'     L---------'     L---------'

                          Diagram 1.



(1) This footnote refers to the diagram.
```

TENEX TECO Editing Example

```
@teco ←——— 1

*;y$

INPUT FILE: ex1.TECO;21 [Confirm] 1204 CHARS| ←— 1

*spaeg$                                   |
                                          | ←— 2
*0lt$                                     |
which must be kept on the same paeg.      |

*<rpaeg$page$0lt>$ ←—————— 3
which must be kept on the same page.
If there is no room for it here on page 1,
then it will occur on page 2.
then it will occur on page 2.

*ht$ ←——— 4
.LEFT MARGIN 10
.RIGHT MARGIN 70
.PAGE SIZE 20
.TITLE ##########^^TENEX Text Processing Example
.SPACING 2
.SKIP 3
.CENTER
#######################A TENEX Text Processing Example
.PARAGRAPH
This is an example of text processing which
shows how text can be inserted in free form
and then formatted by the computer.
.PARAGRAPH
Following this paragraph is a block diagram (1)
which must be kept on the same page.
.FOOTNOTE 3
.CENTER
________________________________
(1) This footnote refers to the diagram.
!
If there is no room for it here on page 1,
then it will occur on page 2.
.SPACING 1
.SKIP 1
.TEST PAGE 10
.NOFILL

          ________     ________     ________
         |        |   |        |   |        |
         | Begin  |-->| Middle |-->|  End   |
         |        |   |        |   |        |
          ________     ________     ________
.FILL
.SKIP 1
.CENTER
*********** Your mother wears combat boots *******
.PAGE
TENEX RUNOFF WILL NOT PRINT FOOTNOTES ON THE LAST PAGE OF A DOCUMENT,
SO IT IS NECESSARY TO GENERATE THIS ^&EXTRA\& PAGE
IN ORDER TO PRINT THE FOOTNOTE.

*

 sYour mothe$                                        |
                                                     | ←— 5
 *0lt$                                               |
 *********** Your mother wears combat boots *******  |

 *k$

 *i#######################Diagram 1. ←——— 6
 $

 *-2l3t$ ←—— 7
 .CENTER
 #######################Diagram 1.
 .PAGE                                        |
                                              |
 *;u$                                         | ←—— 8
                                              |
 OUTPUT FILE: ex1.TECO;22 [New version]       |
 *;h$
 $
```

```
                A TENEX Text Processing Example

     This is an example of text processing which shows how
text can be inserted in free form and then formatted by the
computer.

     Following this paragraph is a block diagram (1) which
must be kept on the same page. If there is no room for it
here on page 1, then it will occur on page 2.

     ---------     ---------     ---------
     |       |     |       |     |       |
     | Begin |---->| Middle|---->|  End  |
     |       |     |       |     |       |
     ---------     ---------     ---------
                  Diagram 1.


(1) this footnote refers to the diagram.


TENEX Text Processing Example              Page   2


TENEX RUNOFF WILL NOT PRINT FOOTNOTES ON THE LAST PAGE OF A
DOCUMENT, SO IT IS NECESSARY TO GENERATE THIS EXTRA PAGE IN
ORDER TO PRINT THE FOOTNOTE.
```

```
                 Multics Runoff Output Example


               A Multics Text Processing Example

     This is an example of text processing which shows how
text can be inserted in free form and then formatted by the
computer.

     Following this paragraph is a block diagram (1) which
must be kept on the same page. If there is no room for it here
on page 1 then it will occur on page 2.

---------------------------------------------------------------
(1) This footnote refers to the diagram.
Multics Text Processing Example                              2

     ---------     ---------     ---------
     |       |     |       |     |       |
     | Begin |---->| Middle|---->|  End  |
     |       |     |       |     |       |
     ---------     ---------     ---------

                   Diagram 1.
```

```
                 Multics QEDX Editing Example

qedx           }<---- 1
r ex2.runoff   }
1,$gp/pgae/<---------> 2
which must be kept on the same pgae.
If there is no room for it here on pgae
.ur %Np% then it will occur on pgae %NNp%.

1,$s/pgae/page/ <------------------3

1,$p<------------------------- 4
.pl 34
.ds
.ce

A Multics Text Processing Example
.he 1 'Multics Text Processing Example''%'

          This is an example of text processing which shows how text can
inserted in free form and then formatted by the computer.

          Following this paragraph is a block diagram
.ft
This footnote refers to the diagram.
.ft
which must be kept on the same page.
If there is no room for it here on page
.ur %Np% then it will occur on page %NNp%.
.sp 2
.ss
.nf 7
.ce 7
.ne 7

---------     ---------     ---------
|       |     |       |     |       |
| Begin |---->| Middle|---->|  End  |
|       |     |       |     |       |
---------     ---------     ---------
.ce
.sp 2
****** If God wanted man to do anything, He'd have done it Himself******
/If God/d<-------- 5
a           }
Diagram 1.  }<------ 6
\34         }
.-1,$p
.sp 2       }<------ 7
Diagram 1.  }
w           }<------ 8
q           }
r 1419  3.213  127+46
```

REFERENCES[1]

Bolt, Beranek, & Newman, Inc. *TENEX user's guide.* Cambridge, Mass: Author, 1973.

International Business Machines Corporation. TEXT/360 reference manual and operating guide. Program No. 360D-29.4.001, 1968.

International Business Machines Corporation. System/360 Administrative Terminal System. Terminal Operations Manual, No. H20-0859, 1972.

Massachusetts Institute of Technology, Project MAC. *Multics programmer's manual.* Cambridge, Mass: Author, 1972.

University of California, San Diego. *EDITOR: User guide to UCSD text editing program for the B6700.* La Jolla, Calif: Author, 1971.

University of Illinois. *PLORTS user guide.* Urbana, Ill: Author, 1973.

University of Michigan. *MTS Volume 1: MTS and the computing center.* Ann Arbor, Mich: Author, 1972.

University of Michigan. *MTS Volume 5: System services.* Ann Arbor, Mich: Author, 1972.

Van Dam, A., & Rice, D. E. On-line text editing: A survey. Computing Surveys, 1971, 3, 93-114.

NOTE

1. The references consist of documentation for programs and systems mentioned in the text, as well as a survey article on text editors. The reader should perhaps check closer to home before ordering a manual from a distant installation, since manuals are often incomprehensible away from their home. They are even more often inaccurate, either at home or abroad.