RESEARCH IN WORD PROCESSING NEWSLETTER

Dr. Bradford A. Morgan and Dr. James M. Schwartz, Editors
South Dakota School of Mines and Technology Rapid City, SD 57701 Phone: (605) 394-2481

IN THIS ISSUE . . .

Volume 4 Number 7 **October 1986**

KEYBOARD ENHANCEMENT MACROS

see page 6

ISSN: 0748-5484. Also indexed by **ERIC** and **INSPEC.** The *Research in Word Processing Newsletter* is published 9 times a year (September through May) by the South Dakota School of Mines and Technology, Rapid City, South Dakota 57701-3995; telephone (605)394-2481. Postage paid at Rapid City, South Dakota. **SUBSCRIPTION:** $15 per year (U.S.); $21 per year (Canada); $27 per year (foreign). Address all subscription inquiries and manuscripts to the Editors, *Research in Word Processing Newsletter,* South Dakota School of Mines and Technology, 501 E. St. Joseph, Rapid City, SD 57701-3995. Please allow four to six weeks for subscription processing. **POSTMASTER:** Send address changes to RWPN, South Dakota School of Mines and Technology, 501 E. St. Joseph, Rapid City, SD 57701-3995.

FinalWord II: Word Processing For a College Writing Program

Gordon P. Thomas

Program:	*FinalWord II* (Version 2.0)
Available from:	Mark of the Unicorn 222 Third Street Cambridge, MA 02142 (617) 576-2760
Price:	$395 $145 on an educational discount (3 or more copies on a university purchase order).
Requires:	PC-DOS or MS-DOS operating system, Version 2.0 or higher; at least 192K of memory.
Recommended:	Up to 512K of memory; two disk drives or a hard disk
Applications:	Full-scale word processing of long documents requiring customized keyboard, menu structure, and formats: scholarly applications, technical writing, college writing courses.

Mark of the Unicorn, who designed *FinalWord II*, has been in the word-processing business for a long time, but their programs—*The FinalWord* (Versions 1.XX) and its predecessors *MINCE* and *SCRIBBLE* (modelled after the mainframe program *EMACS*)—have been best suited to sophisticated users who are most likely doing technical writing. Instead of offering on-screen formatting, like so many other popular programs in business, these programs produce ASCII files that contain formatting commands that are embedded in the text. Before the program prints your files, it formats them using some very sophisticated techniques.

These features are also part of *FinalWord II* (and have even been expanded), but Mark of the Unicorn has succeeded in eliminating some of the clumsier features of *The FinalWord* (Versions 1.XX) so that the program is quite suitable for a college or university that wants to introduce students with no experience with computers to word processing and at the same time provide extremely sophisticated word processing for its faculty. However, an important qualification is in order: before all the users can use the program with ease, some sort of "superuser" will have to spend considerable time installing the program for the needs of these various users.

FinalWord II as it comes out of the box is probably best thought of as only partially installed. With some effort on the part of someone who knows other word processors, has a firm understanding of how DOS works, and knows something about programming, *FinalWord II* will run on any computer that uses a DOS operating system, drive any printer, and take advantage of all the features of that printer. In addition, it can be so greatly modified in terms of the appearance of the menus, command structure, names of the embedded commands, and keyboard layout that users of one version would not even recognize another version.

Formatting Features

Mark of the Unicorn does not think of the program as being only partially installed. They have designed it for the sort of person who could benefit from a program like *Nota Bene* (reviewed in these pages in the January issue): a professional writer with many formatting demands and some experience with other word processors. Many of the formatting features are especially suitable for scholars: the program can, for example, print out an entire book by automatically numbering the chapters, sections, subsections, paragraphs and notes using any format desired; create a complete table of contents; automatically number the pages (printing multi-line headers and footers and allowing the extra space on even and odd pages for the binding margin); use long footnotes that carry over to the next pages (separately numbered from the notes at the end if you wish); use a variety of fonts, including true proportional spacing, automatically selecting the appropriate one for chapter headings, the main text, and footnotes; format the text in as many as five columns, placing footnotes at the bottom of the appropriate column; and generate an index that contains entries in hierarchical levels.

Editing Features

But these features can be used only after a long and careful study of the imposing manual and some tinkering with the default values of the program (all of which is done by editing various text files with which the program comes equipped). Novice users have almost no need of these features; nevertheless, they can make use of the program's best feature: its powerful editor. It is fast and safe: moving from beginning to end of very long files is almost instantaneous, and the text is automatically written to disk when you don't touch the keyboard for longer than 10-50 seconds (the exact time can be adjusted). It is hard to lose text completely. Convenient pop-up menus remind you of the hard-to-remember commands and make it easy to execute those commands; you can give commonly used commands in a single stroke by using the function keys. Paragraphs are automatically reformed on the screen. Text to be deleted is highlighted on the screen and saved in a buffer so that it can be restored in case of a mistake or simply moved to a different place in the file. You can toggle between two different files in one command or display them both on the screen in separate windows and toggle between files in each window. Up to 24 different files can be "open" at once and the screen can display up to six "windows," making it easy to edit different parts of a long document or join together parts of several different documents.

Installation Can Be Elaborate

Novice users could not easily take advantage of these features unless you installed the program with their needs in mind. This brings us to the feature that makes *FinalWord II* unique among the programs with which I am familiar. You can describe exactly what you want each one of the control sequences and the function keys to do by writing macros that are constructed with the program's built-in programming language. It's like having a more complex version of *SuperKey* built in to the word processor. You can set up function keys to perform the most common cursor movements or deletion commands—delete backwards to beginning of the sentence or move forward to end of the paragraph, and so forth. For related commands you can construct your own pop-up menus that can, if you wish, access other menus. You can also create your own error messages; to the user, these are indistinguishable from those the program already provides. In a single keystroke, you can cause complicated formatting commands that need to be embedded in the text to appear on the screen: for example, the string "@value[monthname]@value[day], 19@value[year]" will cause the current date to be spelled out on the printed

copy. By doing some careful thinking about the kind of editing commands and formatting commands that certain users are likely to need, you can relatively easily create formatting commands, a keyboard, and a menu system that make optimum use of your particular hardware.

Other Features

As the date command that I mention above shows, *FinalWord II* has a number of features that depend on built-in connections it has to the operating system. Another command will print out the time that the printing took place, and the current time from the operating system is displayed on the screen and updated every time the full screen display is re-drawn. A more useful feature is the fact that you can give operating system commands from inside the word processor. This allows you to hook up other pieces of software, such as style analysis programs or small utilities, to the editor; you can even design pop-up menus that contain macros that perform certain operations outside the word processor itself. For example, the program lacks a convenient method of counting number of words in a document while you are in the editor (the formatter will, however, count words and display the results on the printed copy). You could easily compensate for this by writing a macro that fits into one of your own menus. This macro would save the file you are currently working on, exit you temporarily from the editor, run a word count utility program, and then return you to the editor. The result does not come instantly, but this feature makes the program easy to adapt to special purposes.

One program that you would not need to hook up to *FinalWord II* is a spelling checker, for *FinalWord II* has its own spelling checker. Although Mark of the Unicorn won't admit it, due to license agreements, the spelling checker is actually *MicroSpell* from Trigram Systems. The lexicon of 70,000 words can be loaded completely into memory (if your memory is 512K or more; otherwise, it loads in four parts), allowing the program to operate with the speed of something like Borland's *Lightning* spelling checker. Unlike *Lightning*, though, *FinalWord II* will dump the lexicon from memory when you exit from the editor at the end of the day. *FinalWord II*'s spelling checker will usually make three very accurate guesses at the correct spelling of any word that it encounters that is not in its lexicon. It is very fast and convenient, but the lexicon does not easily fit on the same disk as the editing and command files, so it is usually necessary to change disks during this operation.

Disadvantages

The program does have a number of disadvantages, especially for those accustomed to a program that provides on-screen formatting. Perhaps the most serious problem is that the program does not show page breaks on the screen, so it is often hard to predict the sorts of decisions about formatting that the program will make at print time. You can run the formatter without printing so that the text scrolls across the screen just as it will appear on the paper, but this is irritating with a long document. Another disadvantage is that the formatting commands can make the text look unnecessarily cluttered, and it is easy to misspell a command or leave out a delimiter that prevents the formatting from occurring. The first part of the manual explains fairly clearly how to use virtually all the features of the program that come installed, but novice users will find it overwhelming. A more serious problem is the second half of the manual, which explains how to use the built-in programming language; unless you have considerable experience in programming, you will find much of this section incomprehensible. Finally, in situations such as a microcomputer lab, where one computer may be shared by several users, serious problems can develop unless the users remember to exit from the program at the end of their sessions. We usually keep our computers running all day, so that students simply put in their disks, load the program, and start typing. When one student fails to exit from the program and another student follows him on the same computer and tries to exit the program before loading it again, the program will cause the directory of the first student's disk to be written on the second student's disk, effectively destroying the data on the second disk. To avoid these problems in a lab setting, you would have to provide careful instructions to the users or use different procedures, such as having each student start up the computer anew each time.

Flexibility Is Its Major Strength

To my mind, though, these disadvantages are outweighed by the program's strong features: a fast and powerful editor, every formatting bell and whistle you could hope for, and extraordinary flexibility in adapting to different hardware and the needs of different users. In a university writing program, for example, two types of users will eventually develop: novices (the students), who need clear menus and a powerful but simple editor, and long-time users (faculty members and graduate students), who probably use their word processors almost daily, often work on long documents, and require very sophisticated word processing. In addition, users of both types often alternate between different types of computers and printers.

In my own department, for example, both students and faculty are well equipped with DEC Rainbows. But individual faculty members may have their own IBM-PCs at home. The students use the DEC equipment while they take courses from us, but when the semester is over they will go on to use other microcomputer labs on campus that are equipped with IBM-PCs or IBM-ATs. Some of these have the standard IBM keyboards; others have Keytronics keyboards. There are approximately four or five different kinds of printers in use by these same people, ranging from dot-matrix ones to laser printers.

This situation is not unusual in other university settings. Each keyboard requires a different arrangement depending on the users' needs; the new laser printers can easily handle five or six fonts; most dot matrix printers can handle the extended ASCII character set. *FinalWord II* can make maximum use of all this hardware and admirably suit the needs of all users except those of middle-level ability who already have their own habits. Installing it to perform all these tasks, however, will not be easy.

Gordon P. Thomas teaches in the English Department at the University of Idaho in Moscow, ID 83843. He can be reached at (208) 855-6384.

NCTE Convention in San Antonio

San Antonio, Texas, will be the site of the 1986 Annual Convention of the National Council of Teachers of English, to be held November 21-26, 1986. Sessions on computer applications to writing include ■ Developing and Implementing a High School Computer-Writing Program ■ Beyond Word Processing: What's Next for the Computer in the Classroom? ■ Computing and Composing: Recent Developments ■ Creative Ways to Use Computers in the English Classroom. At least two one-day workshops will pursue the theme: ■Using Computers to Enrich the Teaching of English ■ The Computer in Your English Class: Effective Applications. Contact NCTE, 1111 Kenyon Road, Urbana, IL 61801.

Word Processing as a Tool for Revision

David F. Noble

Many users of word processing do not know that certain programs and techniques can speed up and improve the process of revision. When you have to meet a deadline, any tools that help you revise more quickly are worth consideration. If, however, they also help you revise better as they shift much of the burden to the computer, they are more than "nice to have" options: they're necessities.

What are these programs and techniques? Word-processing programs that can use macros for specific revision tasks. A *macro* is a large command that strings together many individual commands. To use a macro, you merely press one or two keys to launch a series of commands in which each command takes place in turn without further help.

Some word-processing programs, like *WordPerfect* from WordPerfect Corporation, *MultiMate* from Ashton-Tate, *XyWrite III* from XyQuest Inc., and *WordStar 2000* from MicroPro International Corporation, have their own macro capabilities. Other programs, even MicroPro's powerful and well-established *WordStar*, currently need the help of an external program called a keyboard enhancer or macro processor.

An example of such software is *ProKey* from RoseSoft. This program runs with IBM PC DOS or MS-DOS operating systems, and the manufacturer claims that the program is compatible with about 4,000 other off-the-shelf software programs, including *dBASE III* and *Lotus 1-2-3*. Similar programs are *SmartKey* from Software Research Technologies, *SuperKey* from Borland International, and *Keyworks* from Alpha Software Corp.

You load the keyboard enhancer and your macro file into the computer before you load your word-processing program. You can load each manually, or you can create a batch file that will load all three automatically. For the rest of the session at the keyboard, you can use existing macros and create new ones with ease.

Macros for Transposing Words, Sentences, and Phrases

How can macros speed up revision? A common task in revising drafts is the relocation of misplaced text elements. Words like *only* and *also* are often in the wrong place in a sentence, sentences are sometimes in the wrong order within a paragraph, and particular phrases occasionally are not in their most emphatic position. To solve some problems of misplacement, you can build macros that transpose characters, words, phrases of any length, sentences, and even paragraphs.

A Word Transposer

The following is a *ProKey-WordStar* macro that transposes two words if you place the cursor on the first character of the first word before you start the macro:

◄begdef►◄ctrl2►◄guard►◄ctrlk►b◄ctrlf►◄ctrlk►k◄ctrlf►◄ctrlk◄v►ctrlk►h◄enddef►

In ProKey macros, *begdef* in angle brackets indicates the beginning of the definition of the macro, and *enddef* marks the end. The second designation in angle brackets *(ctrl2)*is the macro name, which you can alter to fit a scheme you devise. When you press and hold down the *Ctrl*(Control) key and press also the *2* key, the commands of the macro take place in turn.

The word *guard* in angle brackets is a *ProKey Version 4.0* option that protects the macro from inadvertent erasure. If you try to redefine *Ctrl-2* with a guard statement in the macro, the computer beeps, and you are asked to confirm that you want to redefine the macro. Pressing *N* returns you to your file. If you press *Y*, you can proceed to redefine the macro.

The following list summarizes the commands in the Ctrl-2 macro:

^KB	Marks and highlights the beginning of the first word as if it were a 'block.' In the operation of the macro, this highlighting is invisible
^F	Makes the cursor jump to the beginning of the second of the two words to be transposed.
^KK	Marks the end of the first word as a block.
^F	Makes the cursor jump to the beginning of the word just after the pair to be transposed.
^KV	Moves the marked word so that it "leapfrogs" the second word.
^KH	Removes the highlighting and completes the transposition.

This macro is simply a series of six commands triggered by a combination of two keystrokes. The commands are performed sequentially in just over 3 seconds. That result is 8 seconds faster than typing the same commands and waiting for each response.

A Sentence Transposer

Transposition macros can become complex if you design them to perform more operations. The following *ProKey-WordStar* macro is a sentence transposer that also reforms the paragraph after the transposition. An assumption behind the design of this macro is that you must first move the cursor to the first character of the first of the two sentences to be transposed.

```
◄begdef►◄ctrl4►◄guard►◄ctrlk►b◄ctrlq►f. ◄enter►
◄enter►
◄ctrlk►k◄ctrlk►1◄ctrlq►f. ◄enter►
◄enter►
◄ctrlk►v◄ctrlk►h◄ctrlq►1◄ctrlb►◄ctrlq►1◄ctrlk►1◄enddef►
```

This guarded macro, named *Ctrl-4*, begins with a *Ctrl-KB* command, which places a beginning-of-block marker at the beginning of the first sentence. The *Ctrl-QF* command initiates a search for a period and a space to find the end of the sentence and to place an end-of-block marker there. (Note that if you customarily put two spaces between sentences, the search string would be a period and two spaces in the first and third lines of the macro.) In a *WordStar* search operation, the cursor comes to rest at the first character position after the search string. The cursor will therefore end up on the first character of the second sentence. The first *Enter* command ends definition of the search string. No options are selected, so the second *Enter* command actually launches the search operation. The *Ctrl-KK* command places the end-of block marker, and the marking of the first sentence is completed. If you were to mark the sentence as a block manually, it would be highlighted at this point, but no highlighting is visible when the macro runs.

Preparation is now made for reforming the paragraph after the transposition. The *Ctrl-K1* command places a K1 marker at the beginning of the second sentence. If you create this macro interactively, having *ProKey* record your keystrokes, be sure to issue the *Ctrl-K1* command by pressing and holding down the *Ctrl* key while you press also the *K* key, releasing both, and then pressing the *1* key. If you hold down the *Ctrl* key while you press also the *1* key, *WordStar* may do strange things.

Another *Ctrl-QF* command begins a search for the end of the second of the two sentences to be transposed, and the two *Enter* commands lead to the initiation of this search operation. The cursor will end up just after the space that follows the second sentence. The *Ctrl-KV* command makes the marked, first sentence leapfrog over the second sentence and so performs the transposition. Highlighting is removed with the *Ctrl-KH* command. That is, the first sentence would be highlighted in its new position if this command weren't issued.

Reforming is next. The *Ctrl-Q1* command takes the cursor back to the *K1* marker, and the *Ctrl-B* command reforms the paragraph from that point to the end of the paragraph. Another *Ctrl-Q1* command takes the cursor one more

time back to the *K1* marker to delete the marker, and the *Ctrl-K1* command as a toggle performs the deletion. The cursor comes to rest at the beginning of the new first sentence and awaits your next act of revision.

It takes only 8 seconds for the macro to do flawlessly a series of tasks that take about 33 seconds to do manually. If you were to make a typing mistake or two, the series would take even longer. This macro can save minutes even if you transpose sentences only several times during revision.

You can see that you don't need to know how to program to create and use macros. You just have to know the commands of your word-processing program. *ProKey* or some other keyboard enhancer can record those commands as you type them interactively and can therefore turn a specific sequence of commands into a macro. You can then use this macro whenever you type its name, such as *Ctrl-4*, the name for the sentence transposer.

A Variable-Length Phrase Transposer

One other transposition macro shows that a macro can be flexible in what it does. It can transpose phrases of any length if you indicate by number interactively how many words are in each phrase. An assumption behind this variable-length phrase transposer, which follows, is that you must place the cursor on the first character of the first word of the first phrase before you start the macro by pressing *Ctrl-3*.

```
◄begdef►◄ctrl3►◄guard►◄ctrlk►b◄ctrlq►f ◄enter►
◄vfld►. . .◄vfld►n◄enter►
◄ctrlk►k◄ctrlq►f ◄enter►
◄vfld►. . .◄vfld►n◄enter►
◄ctrlk►v◄ctrlk►h◄enddef►
```

In broad lines, the macro marks as a block a phrase whose length you specify numerically by indicating the number of words in the phrase and pressing the *Enter* key. The macro soon waits for definition of the second phrase whose length you specify in the same manner. Finally, the macro moves the first phrase as a marked block to the point just after the second phrase and thus completes the transposition.

In the first line of the macro, the *Ctrl-KB* command plants a marker at the beginning of the first phrase and starts the process of marking the phrase as a block. A way is now needed to specify the length of the phrase. The simplest way is to begin a search (*Ctrl-QF*) for the space that follows a word and to make use of *WordStar*'s number option for a search operation.

When you use *WordStar* by itself and define a search string, the program prompts you for options. If you want *WordStar* to search for the *n*th occurrence of the search string, type the number of the occurrence, type next the letter *n* (the number option), and then press *Enter*. *WordStar* will search for the *n*th occurrence of the search string and move the cursor to the first position just after that occurrence.

If you use *WordStar* with *ProKey*, the same task is much simpler, thanks to *ProKey*'s variable field statement (*◄vfld►. . .◄vfld►*) in the second line of the macro. A variable field provides a pause in the macro's operation so that you can specify how many words are in the first phrase to be transposed. The three periods (. . .) between the two *vfld* commands indicate where you supply the number interactively. After you type the appropriate number, you then press the *Enter* key (to mark the end of the variable field), and the operation of the macro is resumed. Pressing the *N* key before the *Enter* key is unnecessary because the macro includes the *n* just after the variable field statement. (Note: If you make the macro interactively, you create a variable field by pressing *Ctrl-hyphen* twice. Consult your *ProKey* manual for specific information on how to create variable fields.)

Macros for Revising Paragraphs

How else can macros aid revision? You can use macros to break apart paragraphs quickly into separate, spaced sentences. You can then better test a problematical paragraph for unity of thought, best sentence order, adequate development, and coherence. These macros might be called **blockbusters**, and their companion macros, **block rebuilders**. The rebuilders reunite the separated sentences after revision.

A Manual Blockbuster and Companion Block Rebuilder

The following example is a simple interactive blockbuster that lets you break apart a paragraph sentence by sentence:

```
◄begdef►◄altb►◄guard►◄ctrlq►a. ◄enter►
.◄ctrln►◄ctrln►◄enter►
n◄enter►
◄enddef►
```

The meaning of each *WordStar* command of the macro is as follows:

^QA	(*Ctrl-Q, A*) Begins a search-and-replace operation.
.[space]	Defines the search string. Again, if you customarily use two spaces between sentences, you should define the search string as *.[space][space]* (using the space bar to create each space).
Enter	Ends definition of the search string.
.^N^N	Defines the replacement string: a period and two hard returns. The macro thus looks for each period and a space (or two spaces) and replaces them with a period and two hard returns to wedge apart a sentence from the preceding one. If your text is double-spaced, you may want to add another ^*N* to the replacement string so that the separation of the sentences is more evident.
Enter	Ends definition of the replacement string.
n	Instructs *WordStar* to perform the replacement immediately without user confirmation. Note that if you don't specify a number, the *n*(umber) option means "Do it now or automatically" in a search- and-replace operation. This form of the option is not available in search operations.
Enter	Ends definition of the options and launches the search-and-replace operation.

Notice that *Ctrl-N*, which performs the function of the *Enter* key, is used instead of the *Enter* key in the definition of the replacement string. If you used the *Enter* key after the period in that string, the definition of the string would end prematurely, and you would not be able to include the second hard return in the string. Word-processing programs that use the *Enter* key to end string definition, but don't have a substitute or delimiter for the *Enter* key, cannot run blockbusters and block rebuilders.

To try out the macro, select from one of your text files any paragraph whose sentences end with a period, or type into a file the following sample paragraph:

> Paragraph development should not be confused with paragraph length. A long paragraph is not necessarily a developed paragraph, nor is a short paragraph by all means an undeveloped one. A paragraph can be long and not develop anything. It can be very long but never climb to a height nor descend to a depth. A long paragraph may just spread out ideas in all directions, like so much milk spilled on the floor. On the other hand, a paragraph can be short and still be developed, elevating thought to a new level of understanding or probing further into the heart of a problem. Development conceived at its best is a category of quality, not quantity, although both are found in some measure in well-constructed paragraphs. (Adapted from *Improve Your Writing with Word Processing*, Indianapolis: Que Corporation, p. 80.)

Place the cursor anywhere in the first sentence of the paragraph to be fragmented and press *Alt-B*, the name of the macro. After the second sentence is separated from the first, press *Ctrl-L* to repeat the operation, and do this as many times as are necessary to break apart the whole paragraph. After all the sentences are separated, move the cursor to the first sentence and press in turn *Ctrl-Q, Q*, and then *Ctrl-B* to reform the separated sentences repeatedly. (If you hold down the *Ctrl* key and press the *Q* key twice, *Ctrl-QCtrl-Q* will appear in the upper left corner of the screen. As is true for most *WordStar* *Ctrl-Q* and *Ctrl-K* commands, the *Ctrl* key needs to be pressed and held down for just the first letter of a dual-letter command.) As soon as the last sentence is reformed, press the space bar to interrupt the reforming process. Your disassembled paragraph should look like this:

Paragraph development should not be confused with paragraph length.

A long paragraph is not necessarily a developed paragraph, nor is a short paragraph by all means an undeveloped one.

A paragraph can be long and not develop anything.

It can be very long but never climb to a height nor descend to a depth.

A long paragraph may just spread out ideas in all directions, like so much milk spilled on the floor.

On the other hand, a paragraph can be short and still be developed, elevating thought to a new level of understanding or probing further into the heart of a problem.

Development conceived at its best is a category of quality, not quantity, although both are found in some measure in well-constructed paragraphs.

The paragraph is now in a form that makes analysis and revision easier. You can better see the paragraph's sentences now that they are distinct from one another. Because the sentences are visible units, you can more easily test their relation to each other and to the whole paragraph, and so test the paragraph's unity of thought.

You can also better perceive the progression of the separated sentences and thereby test their sequential order. If you notice that a sentence is out of place, the blank line between sentences helps you imagine where the displaced sentence may be better located.

Blank lines also help you test a paragraph for adequate development. If the paragraph does not say enough or if you need to expand a sentence or add a sentence or two, the blank lines enable you to picture where to make the expansion or insertion.

Blank lines are an aid as well for testing paragraph coherence. As physical gaps, the spaces accentuate gaps in thought between sentences but seem to disappear if sentences contain appropriate transitional devices and, more important, are tightly linked in meaning to preceding and following sentences.

The companion block rebuilder (*Alt-R*), which you use after you have finished revising the paragraph, is the same as the blockbuster, except that the search and replacement strings are reversed:

◄begdef►◄altr►◄guard►◄ctrlq►a.◄ctrln►◄ctrln►◄enter►
. ◄enter►
n◄enter►
◄enddef►

You use this macro the same way: positioning the cursor anywhere in the first sentence, pressing *Alt-R* to start the macro, and pressing *Ctrl-L* to repeat the operation manually until all the sentences of the paragraph are rejoined. To reform the paragraph, move the cursor to the first line of the paragraph and simply press *Ctrl-B*. Because the sentences are no longer separated, the *Ctrl-QQ Ctrl-B repeat-reform* command is unnecessary.

A Semiautomatic Blockbuster and Block Rebuilder

For greater ease, you can build a *ProKey-WordStar* macro that will pause for you to specify how many sentences are in the paragraph. After you indicate the number and press the *Enter* key, the macro automatically separates the number of sentences you have specified. This macro is a workhorse, and after you use it and its companion block rebuilder several times, you'll wonder how you could ever revise future "difficult" paragraphs without this pair of macros.

In this set the blockbuster, arbitrarily named *Alt-O*, is a seven-line macro:

◄begdef►◄alto►◄guard►◄ctrlq►f◄ctrln►◄ctrln►◄enter
b◄enter►
◄dn►◄dn►◄ctrlk►1◄ctrlq►a. ◄enter►
.◄ctrln►◄ctrln►◄enter►
◄vfld►. . .◄vfld►n◄enter►
◄ctrlq►1◄enter►
◄ctrlq►q◄ctrlb►2◄enddef►

An explanation of each command makes clear the action of this useful macro.

^QF	Begins a *WordStar* search operation. This operation lets you start the macro anywhere in the paragraph. Therefore, you won't have to move the cursor manually to the first sentence, as is necessary for the *Alt-B* blockbuster.
^N^N	Defines the search string, which, in this macro consists of the two hard returns that precede the paragraph. The search is a backward search to find the beginning of the paragraph.
Enter	Ends definition of the search string.
b	Specifies that the search is to be backward.
Enter	Ends specification of the search options.
dn	Moves the cursor down a line and over to the left margin. In a backward search in *WordStar*, the cursor becomes positioned on the first character of the search string. Two down arrows are therefore needed to move the cursor forward to the beginning of the paragraph.
dn	Moves the cursor down another line and to the beginning of the paragraph.
^K1	Plants a K1 marker at the beginning of the paragraph. This marker will make automatic reforming possible after the macro has separated the paragraph's sentences.
^QA	Begins a search-and-replace operation. You will recognize this and the next five steps as being the same as those in the Alt-B blockbuster.
.[space]	Defines the search string. Be certain to press the space bar twice if you customarily use two spaces between sentences.
Enter	Ends definition of the search string.
.^N^N	Defines the replacement string. Remember to use a third *Ctrl-N* if your text is double-spaced.
Enter	Ends definition of the replacement string.
vfld	Begins a variable field. (You press *Ctrl-hyphen* [not the minus key on the keypad] to create a variable field.) This *ProKey* feature, together with the *n* option, gives the macro its magic. A variable field provides a pause in the macro's operation so that you can enter information, which can be any number of characters long. After you finish entering the information, you press the *Enter* key, and the operation of the macro is resumed.
vfld	*(Ctrl-hyphen)* Ends definition of the variable field. Note that three periods (. . .) appear in the macro between the two *vfld* statements. The periods indicate where the user supplies information of any length interactively.
n	Specifies that the replacement should occur the number of times you specify during the pause. For example, if you have a paragraph with five sentences and you want them all separated, press the 5 key and then the *Enter* key to end entry of information. *WordStar* will perform the search-and- replace operation five times and then be receptive to the next instruction in the macro.
Enter	Ends definition of the options and launches the numerically specified search-and-replace operation.
^Q1	Moves the cursor back to the K1 marker to prepare for reforming the separated sentences.

Enter	Positions the first sentence from beside the K1 marker to the line below it. This *Enter* command (which could be *Ctrl-M* instead) is important. Without it, the first line of the paragraph whould be erased when the K1 marker is removed.
^QQ	Provides the first part of a *Ctrl-QQ Ctrl-B repeat-reform* command to reform the separated sentences.
^B	Provides the rest of this command.
2	Sets the reforming process to just below the fastest speed to give you better control over interrupting the process with the space bar after the macro has reformed all the separated sentences. The fastest speed is 1; the slowest, 9. *WordStar*'s default rate is 3.

Before you use this macro, check to make certain that the paragraph ends with a period and a space (or a period and two spaces depending on how you space between sentences). The purpose is to create an additional set of blank lines between paragraphs so that you can differentiate disassembled paragraphs from one another if you want to break apart and revise several paragraphs at the same time.

Start the macro at any point in the paragraph and wait for the pause for you to specify how many sentences the paragraph has. If, like the sample paragraph on paragraph development, this paragraph has seven sentences, press the *7* key and afterward the *Enter* key. Then sit back and wait for the macro to begin reforming the sentences.

When reforming begins, position a finger near the space bar so that you can press it quickly to interrupt the reforming process after the macro has reformed the last separated sentence. As soon as the cursor stops, the disassembled paragraph is ready for analysis and any revision.

The following four-line macro, named Alt-J, is the companion block rebuilder to the Alt-O blockbuster:

```
◄begdef►◄altj►◄guard►◄ctrlq►1◄ctrlq►a.◄ctrln►◄ctrln►◄enter►
. ◄enter►
◄vfld►. . .◄vfld►n◄enter►
◄ctrlq►1◄ctrlk►1◄ctrly►◄ctrlb►◄ctrlx►◄enddef►
```

As is true for any block-rebuilding macro, the search and replacement strings are the reverse of those of the companion blockbuster. Like the Alt-O blockbuster, this Alt-J block rebuilder has a variable field for specifying the number of sentences to be rejoined after revision. If you have divided, added, or deleted sentences during revision, the number you specify will probably be different from what you indicated when you used the blockbuster to break up the paragraph.

A novel feature in the Alt-J block rebuilder is the *Ctrl-Q1* command in the first line of the macro. The command moves the cursor back to the *K1* marker, placed by the Alt-O macro at the beginning of the paragraph. Because of that marker, you can start the Alt-J block rebuilder anywhere in the paragraph (or from anywhere else in the file, for that matter). Make certain, therefore, that you don't delete the *K1* marker during your revision of the paragraph.

The *Ctrl-Q1*command in the last line of the Alt-J block rebuilder sends the cursor back to the *K1* marker one more time to delete (by the *Ctrl-K1* command) the marker and (by the *Ctrl-Y* command) the line the marker occupies. You can see now the reason for the *Enter* command near the end of the Alt-O macro. That command shunts the first line of the paragraph to a new line so that no text is lost with the deletion of the *K1* marker by the Alt-J macro.

Like the Alt-R block rebuilder, the Alt-J macro has just a *Ctrl-B* command for reforming the reassembled paragraph. After it is reformed, the *Ctrl-X* command is an extra touch that moves the cursor down a line to the beginning of the next paragraph.

The Alt-O and Alt-J macros show how paragraph manipulation can be somewhat automated. If you want, you can build a large macro that will take apart all paragraphs automatically throughout a whole file, whether it is a

500-word theme or a 20-page research paper. (See, for example, the Alt-F macro in *Improve Your Writing with Word Processing*.) Of course, a long paper will take longer to process, but the program can handle this or longer texts with no more effort on your part.

Macros are fairly simple when they are to separate sentences that end with periods only. If, however, the task is to separate sentences with different end punctuation, macros can become long and complex. For these, you must build into the macro a series of loops to handle each kind of end punctuation. Each loop lengthens the action of the macro but also makes it more useful. You can tailor your macros for the different end punctuation you use.

Macros for Indenting

After you have disassembled a paragraph into separate sentences, you can invoke other macros to indent sentences subordinate to preceding ones. This strategy is helpful when you are revising a long paragraph and want to understand better its structure—that is, the order and relation of the sentences to each other and to the whole paragraph.

Some sentences offer a minor comment to a major assertion, an elaboration of a main point, or an extension of a preceding idea. When you indent subordinate sentences beneath dominant ones, you can quickly size up the shape of your paragraph and make better judgments about its design and direction of thought. You can make macros to create several levels of subordination if you customarily work with long and complex paragraphs. The following are examples of indentation macros:

◄begdef►◄alttab►◄ctrlo►l6◄enter►◄ctrlb►◄ctrlx►◄enddef►
◄begdef►◄ctrltab►◄ctrlo►l11◄enter►◄ctrlb►◄ctrlx►◄enddef►
◄begdef►◄ctrl/►(ctrlo►l16◄enter►◄ctrlb►◄ctrlx►◄enddef►

The Alt-Tab macro indents text by resetting the left margin at the sixth character position, and the Ctrl-Tab and Ctrl-/ macros reset the left margin at the eleventh and sixteenth positions, respectively. The *OL* command calls for resetting the left margin, and the number after the command indicates the new margin position. The *Enter* command activates the resetting process.

Two additional commands make each macro more useful. A *Ctrl-B* command reforms the first separated sentence. For convenience, a *Ctrl-X* command moves the cursor to the first line of the next separated sentence so that it, too, can be reformed.

If these macros were used to indent the separated sentences of the sample paragraph on paragraph development, the disassembled paragraph would look like this:

> Paragraph development should not be confused with paragraph length.
>
> A long paragraph is not necessarily a developed paragraph, nor is a short paragraph by all means an undeveloped one.
>
> > A paragraph can be long and not develop anything.
> >
> > > It can be very long but never climb to a height nor descend to a depth.
> > >
> > > A long paragraph may just spread out ideas in all directions, like so much milk spilled on the floor.
> >
> > On the other hand, a paragraph can be short and still be developed, elevating thought to a new level of understanding or probing further into the heart of a problem.
>
> Development conceived at its best is a category of quality, not quantity, although both are found in some measure in well-constructed paragraphs.

Indenting separated sentences to display various levels of subordination helps you see the design or shape of a paragraph. If a sentence for balance is missing, its absence will be more noticeable as you examine subordination in your disassembled paragraphs. This kind of structural testing is easy to do with these indentation macros.

After you are through with these macros, you can use the following macro to restore the left margin to the first character position:

◄*begdef*►◄*alt*►◄*guard*►◄*ctrlo*►*l1*◄*enter*►◄*ctrlb*►◄*ctrlx*►◄*enddef*►

As you can see, this macro is like the indentation macros, except for the number 1, which reestablishes the left margin at the first character position.

Macros for Revising Sentences

You can even build sentence-splitting macros to break apart long, awkward sentences for analysis and revision. Often awkwardness arises from faulty parallelism. You are better able to detect this stylistic weakness when you break down the clumsy sentence where it is punctuated internally.

Companion **sentence restorers** rebuild dismantled sentences after revision. You can design these macros for sentences that blockbusters have separated and so integrate the revision of paragraphs and sentences.

Simple macros for sentence-splitting use only commas for targets because most sentences have just commas for internal punctuation. Elaborate **sentence splitters** loop several times to handle different kinds of internal punctuation, like semicolons, colons, and dashes. Again, you can tailor these macros to include the kinds of internal punctuation you normally find in the text you write and revise.

The following macro is an interactive sentence splitter that breaks apart a sentence wherever it has a comma. The macro is designed to split sentences of disassembled paragraphs and to be started anywhere within a separated sentence.

◄*begdef*►◄*alt4*►◄*guard*►◄*ctrlq*►*f*◄*ctrln*►◄*ctrln*►◄*enter*►
b◄*enter*►
◄*ctrlx*►◄*ctrlk*►*2*◄*ctrlq*►*a,* ◄*enter*►
, ◄*ctrln*►◄*ctrln*►◄*enter*►
◄*vfld*►*. . .*◄*vfld*►*n*◄*enter*►
◄*ctrlq*►*2*◄*ctrlx*►◄*ctrlq*►*q*◄*ctrlb*►*2*◄*enddef*►

This macro searches backward for two *Ctrl-N*'s; moves the cursor down a line to the left margin (*Ctrl-X*); plants a *K2* marker; searches (*Ctrl-QA*) for a comma and a space; replaces a found string with a comma, a space, and two *Ctrl-N*'s; does this the number of times you specify interactively (◄*vfld*►*. . .*◄*vfld*►*n*); returns to the *K2* marker (*Ctrl-Q2*); drops down a line (*Ctrl-X*); and repeat-reforms (*Ctrl-QQ Ctrl-B*) at a controllable rate (2 in *WordStar*'s range from 1 to 9) the separated words, phrases, or clauses.

The *K2* marker is used so that this macro can operate on sentences separated by the Alt-O macro, which uses the *K1* marker. Accordingly, the *Ctrl-Q2* command is needed to take the cursor back to the *K2* marker.

Note that the Alt-4 sentence splitter will work with separated sentences even after they have been indented by indentation macros. This capability makes the macro particularly useful for analyzing complex sentences within long paragraphs or separately.

Suppose, for example, that you want to analyze the opening sentence of Thomas Jefferson's **First Inaugural Address**:

> Called upon to undertake the duties of the first Executive office of our country, I avail myself of the presence of that portion of my fellow citizens which is here assembled to express my grateful thanks for the favor with which they have been pleased to look towards me, to declare a sincere consciousness that the task is above my talents, & that I approach it with those anxious & awful presentiments, which the greatness of the charge, & the weakness of my powers so justly inspire.

After you run the Alt-4 sentence splitter, the split-up sentence would look like this:

> Called upon to undertake the duties of the first Executive office of our country,
>
> I avail myself of the presence of that portion of my fellow citizens which is here assembled to express my grateful thanks for the favor with which they have been pleased to look towards me,
>
> to declare a sincere consciousness that the task is above my talents,
>
> & that I approach it with those anxious & awful presentiments,
>
> which the greatness of the charge,
>
> & the weakness of my powers so justly inspire.

Next, with the use of indentation macros and vertical alignment, you could make parallelism in the sentence more evident and so comprehend better its meaning and flow of thought:

> Called upon to undertake the duties of the first Executive office of our country,
>
> I avail myself of the presence of that portion of my fellow citizens which is here assembled
>
> to express my grateful thanks for the favor with which they have been pleased to look towards me,
>
> to declare a sincere consciousness that the task is above my talents,
>
> & that I approach it with those anxious & awful presentiments,
>
> which the greatness of the charge,
>
> & the weakness of my powers so justly inspire.

After you have analyzed the sentence (and revised it if it were your sentence and you were unhappy with it), you could quickly delete the tab spacing in each indented line and then run the following commas-only sentence restorer to put the sentence back together:

```
◄begdef►◄alt5►◄guard►◄ctrlq►2◄ctrlq)a, ◄ctrln►◄ctrln►◄enter►
, ◄enter►
◄vfld►. . .◄vfld►n◄enter►
◄ctrlq►2◄ctrlk►2◄ctrlx►◄ctrlb►◄enddef►
```

The first *Ctrl-Q2* command moves the cursor to the *K2* marker, and the numerically specified search-and-replace operation is the reverse of that in the *Alt-4* macro. The second *Ctrl-Q2* command takes the cursor once again to the *K2* marker to delete it and to position the cursor for reforming the rebuilt sentence.

Besides macros for transposing, blockbusting, indenting, and sentence splitting, you can build other kinds of macros that will simplify the processes of saving a file to prevent the loss of work; marking and moving words, sentences, lines, and paragraphs; reforming paragraphs if your word-processing program doesn't do this

automatically; locating sentence beginnings quickly for revision; and adding or deleting hard returns. You can make a macro for almost all the word-processing tasks you do when you write and revise. Any series of keystrokes can become a macro you can store and use again in future sessions.

If you have a word-processing program that can use macros, it can be a powerful and efficient tool to help you perfect your work. Therefore, if you are planning to buy a word-processing program, be certain that the package can make its own macros or run with a macro-making program like *ProKey*. Once you know how to create and use macros with word processing, you will discover for yourself that macro-assisted revision is an indispensable strategy for better writing.

David F. Noble, Ph.D., is Editorial Director at Que Corporation, a publisher of computer books for the IBM Personal Computer and compatibles. Before joining Que, he taught writing and literature for 16 years at Indiana Central University (now the University of Indianapolis). He is coauthor of *Improve Your Writing with Word Processing* (Indianapolis: Que Corporation, 1984), which presents over 100 macros to aid writing and revision. About 65 of these are *ProKey* macros for *WordStar*. He has written also the article on "Word Processing" for the latest edition of the *Encyclopedia Americana*. Comments and dialogue are welcome; contact David Noble at Que Corporation, 7999 Knue Road, Indianapolis, IN 46250.

The Shakespeare Data Bank

At present, 128 scholars around the world have become associates of The Shakespeare Data Bank, a project designed "to accurately compile, coordinate, and *condense* into listed sentence form for easy reference by computer as much as possible of what has been written about Shakespeare, his works, and the necessary background," according to Louis Marder, editor of *The Shakespeare Newsletter*. "If it has been printed or delivered as a paper, it will be available in the SDB via a personal or library computer. Indexes, menus, and multiple, suggestive cross-references will simplify the process, offering as little or as much as is desired." Contact Louis Marder, *The Shakespeare Newsletter*, 1217 Ashland Avenue, Evanston, IL 60202, or call (312) 475-7550.

Call for Manuscripts

Published by Oxford University Press, the quarterly *Literary and Linguistic Computing* is the journal of the Association for Literary and Linguistic Computing. Editors of the journal have called for papers: "Proposals are sought for papers dealing with the results of research into language and literature computer applications. Papers that deal with hardware and software, computer-assisted language learning, word processing for humanities and the teaching of computer techniques to language and literature students are also appropriate. Also being sought are survey papers that introduce computing techniques or papers that cover linguistic methodologies, new applications, work in progress and book reviews." Guidelines for contributers can be obtained from Gordon Dixon, Editor-in-Chief, *Literary and Linguistic Computing*, Institute for Advanced Studies, Manchester Polytechnic, All Saints, Manchester, England M15 6BH.

Bibliography Update

Bradford A. Morgan

Anderson, C. W. and G. E. McMaster. "Modeling Emotional Tone in Stories Using Tension Levels and Categorical States." *Computers and the Humanities*. 20:1 (January-March 1986), pp. 3-9.

Associates, I. A. *Manuals That Work: A Practical Guide to Creating Effective, User-friendly Manuals*. Columbia, MD: GP Courseware, 1986. (with videotape)

Bell, Elizabeth S. "On the Value of Subjective Responses: Computer Technology in the Composition Classroom." *The Computer-Assisted Composition Journal*. 1:1 (Summer 1986), pp.4-11.

Bell, Norman T. and James W. Warner. "Desktop Publishing: A New Frontier for Instructional Technologists." *T.H.E. Journal*. 14:1 (August 1986), pp. 71-73.

Bigelow, George. "Word Count: Use This Simple Technique To Count the Number of Words in Any Sequential Text File." *Nibble*. 7:9 (September 1986), pp. 94-95.

Bridges, Linda. "Desktop Publishing Not Expected To Be Widely Accepted Until 1988." *PC Week*. 3:19 (May 13, 1986), pp. 1, 8.

Celko, Joe. "The ABCs of H&J: Whether You Use an Automatic Hyphenation Program, Set Type Ragged, or Go In and Hyphenate Yourself, There Are Many Factors To Consider." *Personal Publishing*. 2:8 (August 1986), pp. 18-21.

Collins, Terence and Lynda Price. "A Guide to Selecting Word-Processing Software for Learning-Disabled College Writers." *The Computer-Assisted Composition Journal*. 1:1 (Summer 1986), pp.12-22.

Couillard, Ted. "Computer-Assisted Composition: What Computers Are Doing for Students at Georgia Southwestern College." *The Computer-Assisted Composition Journal*. 1:1 (Summer 1986), pp.23-27.

Davis, Kevin. "Initiating the Uninitiated: Word-Processing Tutorials for Rapid Results." *The Computer-Assisted Composition Journal*. 1:1 (Summer 1986), pp.28-33.

Dunn-Lardeau, Brenda. "Stylistics and the Computer: Classroom Mix and Match?" *Computers and the Humanities*. 20:1 (January-March 1986), pp. 51-54.

Eger, Henrik. "Discourse and Word-Processing Genes: Thoughts of an Unborn Computer-Assisted Composition Journal." *The Computer-Assisted Composition Journal*. 1:1 (Summer 1986), pp.1-3.

Fasenmyer, Joy Cloy. "Word Processing Conversion Software Might Answer Your Prayers." *The Lawyer's PC*. 3:24 (August 15, 1986), pp.1-5.

Felici, James and Walter Omstead. "MagnaType: The Personal Typographer. More Than a Personal Publishing Program, MagnaType Is a Powerful Front-end Typographic Program That Can Control a Phototypesetter As Easily As It Drives a Laser Printer—Without the Cost of a Minicomputer." *PC World*. 4:7 (July 1986), pp. 202-209.

Frisina, Warren. "Learning Disability and Computers: Beyond Word Processing." *Closing the Gap*. 5:3 (August/September 1986), pp. 22-24.

Goodman, Danny. "Taking a Page from the Pros: PageMaker and Ventura Publisher Will Soon Bring Sophisticated Page Makeup Power to the PC Environment. Both Programs Work Hard at Taming PC Hardware for This Highly Graphic Application, But While They Look Very Similar, Each Is Suited to a Different Style of Document." *PC World*. 4:7 (July1986), pp. 244-251.

Gray, J. C. "Creating the Electronic New Oxford Dictionary." *Computers and the Humanities*. 20:1 (January-March 1986), pp.45-49.

Hamby, Paul W. "In-House Publishing with ClickArt." *The Lawyer's PC*. 3:24 (August 15, 1986), pp.5-10.

Hart, Roger. "Postscript: The Language Used on Digital's New Printserver 40 Laser Printer. PostScript Provides a New Way of Processing and Transmitting Information to Printers and Typesetters." *Hardcopy*. 6:9 (September 1986), pp. 157-159.

Heid, Jim. "The Scenic Route: With a Multitude of Commands and an Arcane Operating System, ScenicWriter Isn't Everyone's Personal Publishing System. Its Power Rivals That of Sophisticated Typesetting Systems, However, Making It the Program of Choice for Demanding Publishing Projects." *PC World*. 4:7 (July 1986), pp. 210-217.

Kimball, Sue L. "Idea Processing with 'ThinkTank' in Advanced Composition." *The Computer-Assisted Composition Journal*. 1:1 (Summer 1986), pp.34-41.

Langman, Larry. *An Illustrated Dictionary of Word Processing*. Phoenix, AZ: Oryx Press, 1986.

McCarty, Willard. "T³, A Multilingual Word Processing Package for the IBM PC." *Computers and the Humanities*. 20:1 (January-March 1986), pp. 57-62.

————. "Nota Bene, an Academic Word Processing and Text Retrieval System for the IBM PC." *Computers and the Humanities*. 20:1 (January-March 1986), pp. 62-71.

McNeill, Dan. "Write On!: Thumbnail Sketches and a Comprehensive Chart of Word Processing Programs for the Apple II Series." *A+: The Independent Guide to Apple Computing*. 4:10 (October 1986), pp.20-27, 31-36.

Meade, Jim. "Finding A Better Way To Handle Corporate Publishing: Digital's Pilot Program in Electronic Publishing Has Wide-Ranging Implications for Digital and for the Growing Electronic Publishing Industry." *Hardcopy*. 6:9 (September 1986), pp. 75-80.

Meeker, Michael W. "Waiting for WANDAH: A Critique of Present Trends in Computer-Assisted Composition." *The Computer-Assisted Composition Journal*. 1:1 (Summer 1986), pp.42-54.

Mellown, Elgin W. "'The Use of the Computer in Literary Studies': An Experimental Course." *The Computer-Assisted Composition Journal*. 1:1 (Summer 1986), pp.55-61.

Mikiewicz, Henry M. "Technological Advances That Have Affected the Printed Word: Historically, the Computer Is the Only Device That Allows Typographers Complete Control Over Their Art." *Hardcopy*. 6:9 (September 1986), pp. 124-130.

Morton, A. Q. "Once. A Test of Authorship Based on Words Which Are Not Repeated in the Sample." *Literary and Linguistic Computing*. 1:1 (1986), pp. 1-8.

Parham, Charles. "Conquering the Dreaded Blank Page: Here's a Look at Prewriting: What It Is and How a Computer Can Help Us Do It." *Computer Classroom Learning*. 7:1 (September 1986), pp.39-44.

Penner, Ken. "The Translator: Use This Flexibile Database Program To Translate Words, Phrases and Sentences from One Language to Another and Back Again. There's Room for Hundreds of Words, Definitions and Notes." *Nibble*. 7:9 (September 1986), pp. 56-67.

Phillips, Charles and Elizabeth Braswell. "Integrating Instruction in Computer Skills and Paragraph Organization." *The Computer-Assisted Composition Journal*. 1:1 (Summer 1986), pp.62-71.

"Pupils at Demonstration School Make News with Publishing Software." *T.H.E. Journal*. 14:1 (August 1986), pp. 50-53.

Resch, Kenneth. "Breaking the Silence Barrier: The Freedom of Word Processing." *Closing the Gap*. 5:3 (August/September 1986), pp. 14-19.

Rodrigues, Dawn and Raymond J. "Teaching Writing with a Word Processor: Grades 7-13." (ERIC No. 52414-016), 1986.

Romesburg, Paul. "Using 1-2-3 As a Word Processor: Why Buy Word Processing Software When You Already Own 1-2-3? Here's How To Use the Spreadsheet Program for Word Processing." *IBM PC Update*. 3:9 (September 1986), pp. 15-18.

Roth, Steve. "Search and Replace: When It Comes Time To Move Files from Program to Program, and To Prepare Copy for a Typesetter or Desktop Publishing System, Substitution Routines Can Be Real Time Savers." *Personal Publishing*. 2:8 (August 1986), pp. 22-23, 26-27.

Sadler, Lynn Veach and Wendy Tibbetts Greene. "The Computer-Assisted Composition Movement." *The Computer-Assisted Composition Journal*. 1:1 (Summer 1986), pp. i-xi.

Selfe, Cynthia L. *Computer-Assisted Instruction in Composition: Create Your Own!* Urbana, IL: National Council of Teachers of English, 1985.

Sharif, Jonaid. "Pre-Writing Aids: Some Computer-Assisted Paths for Invention." *The Computer-Assisted Composition Journal*. 1:1 (Summer 1986), pp.72-80.

Tomlinson, David. "The Borzoi College Writer and PC-Write 2.6: A Review." *The Computer-Assisted Composition Journal*. 1:1 (Summer 1986), pp.81-82.

Ulick, Terry. "Hard Copy Screens: The Power to Reproduce IBM PC Screens Has Been Needed by Many, But Until Recently, You Could Only Take a Picture Or Use the Limited 'Print-Screen.' Now There's a Better Way." *Personal Publishing*. 2:8 (August 1986), pp. 30-31.

Vernot, David. "Best Bets in Shareware for Word Processing." *Electronic Learning*. 6:1 (September 1986), pp. 59-60.

"WordPerfect Is Overwhelming Choice Among Newsletter's Consultants." *The Lawyer's PC*. 3:23 (August 1, 1986), pp. 1-3.

Wresch, William. *A Practical Guide to Computer Uses in the English/Language Arts Classroom*. Englewood Cliffs, NJ: Prentice-Hall, 1986.