A Portable Screen-Oriented Editor

Edward K. Ream

Edward K. Ream, 1850 Summit Ave., Madison, WI 53705.

Are you tired of using a miserable line-oriented editor? Are you unwilling to pay several hundred dollars for a full-screen editor? Are you worried that your text editor will become obsolete if you change machines or operating systems? Are you frustrated that you can't make even simple changes to your editor? Are you unable to adapt your editor for more than one type of terminal? If so, here is an editor for you.

In the past five years I have used several different text editors on my computer system at home. My first editor was a typical character-oriented text editor. This editor did have some good points, but most of the time it forced me to type a lot more than I wanted to. Also, this editor was unsafe to use. Some mistakes that I could easily make caused the editor to lose many lines of text.

My second editor had more potential. It allowed me to do simple things simply. There were easy-to-use commands to move the cursor along a line, and to insert and delete characters on a line. The editor was also safe. Dangerous commands took more typing than harmless commands so editing was much less nerve wracking. I never destroyed more than a few moments of work with this editor.

This editor did have one major drawback: it didn't use the capabilities of the video screen I was using. As a rule, only one line of text was visible on the screen at a time and that line was always on the bottom of the screen. So while the editor had all the commands needed to edit files quickly, in practice the editor was difficult to use. I kept wanting to see the lines above and below the line where I was in order to make sure that I was making changes in the proper context.

I basically liked the editor so I modified it slightly so the screen would show several lines above the line being edited. This modification was actually a very small one; the editor still wasn't perfectly adapted to video displays (the cursor still couldn't move off the bottom line), but the editor was almost as easy to use as a full-screen editor.

Of course, I made other adjustments to that editor. I became fascinated with changing the editor's commands. I noticed a ripple effect; a change in the design of one command might lead me to modify five or six other commands. I spent many pleasant hours just thinking about how commands are interrelated.

After several months, I thought I had a pretty good editor. It was safe. It was predictable. Although complicated editing tasks sometimes took considerable work, simple editing jobs were very simple.

This editor had only one flaw, but that turned out to be fatal. It was written in assembly language. The editor worked fine until I got a new set of disks and a new operating system (CP/M). It soon became woefully apparent that almost 50% of the editor would have to be rewritten in order to transport it from the old operating system to the new. It looked as though I would be spending many weeks on this unpleasant project.

As luck would have it an alternative presented itself just at this time. *Dr. Dobb's Journal* published two articles by Ron Cain in which he presented a compiler for a language he called Small-C. (See *DDJ* #45 and #48, May and September, 1980.) Small-C is a subset of the full langauge called C, one of the best languages for writing systems programs like compilers and editors. Small-C would be perfect as a language in which to write a new editor. Just as importantly, the compiler for Small-C was written in Small-C itself. Although it may not be immediately obvious, that meant that the compiler formed the basis of a completely portable computer system. All programs written in Small-C could be moved from one machine to another just by changing a few routines in the compiler. The compiler was in the public domain, which meant I could use it or modify it however I wanted to. Clearly, the Small-C compiler was a great tool.

I obtained an excellent version of this compiler for the CP/M operating system from a company called The Code Works (TM). I recommend this compiler. It is well documented; it works reliably; all the source for the compiler is provided; and it's all in the public domain. At $17 it is one of the best bargains around.

After becoming acquainted with this compiler, I started work on what I hope will be the last version of this editor. Remember that the only thing wrong with my previous editor was that I couldn't move it easily to a new operating system. Thus, portability became a top priority. I wanted this editor to adapt easily to new operating systems or new display devices.

The Small-C editor shares many features with the old assembly language editor. I saw no reason to change a successful design. As a result, the new editor is as simple to use and reliable as the old editor was. However, the new editor has been completely rewritten. This has allowed me to improve many of the internal details of the editor's operation. I have also added full-screen-editing capability and a prompt line.

As far as I can tell the new editor is very portable. It has been used on widely differing terminals without making any programming changes. The editor comes with a configuration program which asks questions about your terminal. The configuration program then automatically produces two files that tailor the editor to your particular keyboard and display screen. I have not moved the editor to any other operating system besides CP/M, but I don't expect great problems in doing so. All code that depends on a particular operating system has been isolated to a few routines all located on one file. Moving to a new system requires only that these few routines be rewritten.

Basic Features of the Screen Editor

I hope by now you are eager to learn what this editor can do. Let's take a closer look at it. We will start by talking about some of the basic concepts and terms that are used throughout to describe the editor.

The screen editor is comprised of several *modes*. You can switch between modes, but the editor can't be in two modes at the same time. The editor behaves differently in each mode, so you use each mode to do a particular type of editing.

There are three modes: edit mode, command mode and insert mode. You use edit mode for making small changes in many different lines. Command mode is used for making larger (hence potentially more dangerous) changes to the whole file you are editing. You must be in command mode to do anything that will change files on the disk. You use insert mode for making a series of insertions into the text. First drafts are often entered in this mode.

You use *commands* to make things happen in each mode. Which commands you can use depends on what mode you

are in. Do not confuse the terms "commands" and "command mode." Every mode has a set of commands which you can use in that mode. Command mode has its own set of commands.

Special characters are one-letter commands. Special characters must be control keys, so you can use these commands in places where the editor might not otherwise expect a command.

The *prompt line* is the top line of the screen. It tells you things like what line you are editing and what mode you are in. If you get confused about what mode you are in, you can always glance at the prompt line and get your bearings.

Figure 1 shows what the prompt line looks like. The line field tells what line of the file you are currently editing. The column field tells what column on the screen the cursor is on. The column field mainly helps you align columns of text correctly. The next field tells you the name of the file you are currently editing. This field is set and used by several command-mode commands which will be discussed later on. The last field tells you what mode you are in.

Several edit-mode commands consist of two or more letters. These edit-mode commands are called *extended commands*. After you type the first letter of such a command, but before you type the last letter, the prompt line will indicate that you must type one more letter to finish the extended command. For example, after you type an "s" to enter the edit-mode search command the prompt line's mode field will contain "edit: search".

The *cursor* is a distinctive character on the screen. On most video terminals this marker is a character shown with reverse video. In command mode you always type commands on the bottom line so the cursor is always on the bottom line. In edit mode and insert mode the cursor is always on the current line.

The *current line* is the only line you can edit. In command mode the current line may not be shown on the screen, but in edit and insert modes the current line is shown and it contains the cursor. If you want to make a change to a line you must first move the cursor to that line.

The *buffer* is a part of the computer's storage which holds the file you are editing. In order to edit a disk file, you must first read the disk file into the buffer using the load command. As you edit the file, the changes you make are made only to the buffer, not to your disk file.

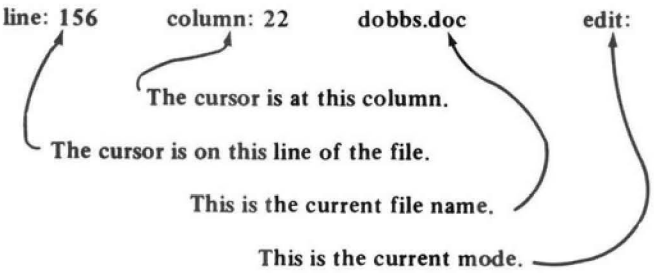

Figure 1: The Prompt Line
The prompt line tells where the cursor is and what the editor is doing.

Before you end your editing session you must copy the buffer back to a disk file using the save or resave commands. The editor keeps track of whether you have done this; it won't let you forget to save your work.

The *window* is a part of the screen which shows a portion of the buffer. In edit and insert modes the window fills all of the screen except the prompt line. You can think of the file as a scroll of parchment which unwinds under the window as you move the cursor. The window is not shown in command mode.

Now let's look more closely at the editor's commands. We'll start with special characters, which are commands that may be used in either insert or edit modes. Then we will look at other commands which are unique to a particular mode.

Special Characters

Special characters act the same in all modes. Special characters must be control characters so that the editor may always distinguish special characters from regular text.

There are 14 special characters; all 14 may be used in either edit mode or insert mode. Only a few may be used in command mode. Figure 2 shows how to switch between modes using three of these keys. Figure 3 summarizes what all these special keys do.

You can use the configuration program to choose which keys on your keyboard to use for each special character. Thus, I won't be able to be specific about what key on your keyboard you will actually hit in order to type, for instance, the down key. I'll often refer to special characters as special keys; for example, I'll call the down special character the down key.

The *left and right keys* just move the cursor. They do not cause changes to lines. These keys are always "anchored" to the current line; you can't move off the current line by using these keys. If you hit the right key while the cursor is at the right margin, nothing will happen. Similarly, nothing will happen if you hit the left key while the cursor is at the left margin.

The *up and down keys* move the cursor up and down on the screen. You can't move the cursor above line 1 of the file, nor can you move the cursor below the last line of the file. Both these keys also switch the editor to edit mode.

The *insert up key* inserts a blank line above the current line while the *insert down key* inserts a blank line below the current line. Both these keys also switch the editor to insert mode.

The *delete character key* deletes the character to the left of the cursor. This key is anchored to the current line; nothing is deleted if the cursor is at the left margin. The *delete line key* deletes the entire line on which the cursor rests.

The *undo key* always undoes whatever editing you have done on the current line since the last time the cursor came to the current line.

The *insert key* switches the editor to insert mode. Similarly, the *command key* switches the editor to command mode and the *edit key* switches the editor to edit mode.

The *split key* splits the current line into two pieces. Everything to the left of the cursor stays right where it is. All other characters are moved from the current line. Both these keys also switch the editor to insert mode.

The *join key* is the opposite of the split key; it combines two lines into one line. The join key appends the current line to the line above it, then deletes the lower line.

Edit Mode

Edit-mode commands are normal letters; you don't need to use special keys. This speeds typing. You can, however, use the special keys in edit mode if you want to.

Besides the special characters you can use the following one-letter edit-mode commands: space, b, c, d, e, g, i, k, s, u, and x. These letters stand for begin, command, down, end, go, insert, kill, search, up, and eXchange. Note that all these commands may be typed either in upper case or in lower case. Letters which are neither edit-mode commands nor special keys are simply ignored. Figure 4 is a brief summary of these commands. The following paragraphs explain what these commands do in greater detail.

The *space bar* moves the cursor right one column. Nothing happens if the cursor is already on the rightmost column of the screen.

The *b command* puts the cursor at the beginning (left-hand margin) of the line.

The *c command* switches the editor to command mode.

The *d command* causes the cursor to move down rapidly. The cursor keeps moving until it reaches the last line of the file or until you type any key.

The *e command* moves the cursor to the right end of the line.

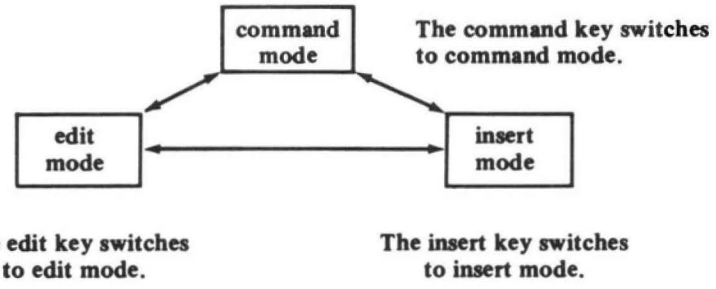

Figure 2: The Modes of the Editor
The edit, insert and command special characters switch the editor from mode to mode.

Special Character	Function
up key:	move cursor up 1 row & enter edit mode
down key:	move cursor down 1 row & enter edit mode
*left key:	move cursor left 1 character
right key:	move cursor right 1 character
insert up key:	insert line above current line & enter insert mode
insert down key:	insert line below current line & enter insert mode
*delete character key:	delete character to left of the cursor
*delete line key:	delete the current line
*undo key:	undo all changes to the current line
*insert key:	enter insert mode
*edit key:	enter edit mode
command key:	enter command mode
split key:	split the current line into two lines
join key:	combine two lines into one line

Figure 3: Special Characters
Special characters are one-character commands that are valid in edit mode and insert mode. Special characters which may be used in command mode are indicated by an asterisk (*).

The *i command* switches the editor to insert mode.

The *g command* moves the cursor to another line. The g command is an extended command; after you type the g, the cursor will move to the prompt line. The prompt line will show "edit: goto:" Now type a line number followed by a carriage return. The cursor will move to the line whose number you typed. If you do not type a valid number the g command does nothing. Leading blanks or minus signs are not allowed. The cursor will move to the last line of the buffer if the number you type is larger than the number of lines in the buffer.

The *k command* is a two-letter command. The second letter you type (the first character after the k) is a search character. The k command deletes from the cursor up to but not including the next occurence of the search character to the right of the cursor. Everything from the cursor to the end of the line is deleted if the search character does not appear to the right of the cursor on the current line. After you hit the k and before you hit the search character the prompt line will show "edit: kill". If you wish to cancel the k command you can hit any control character. The k command will be stopped and no deletion will be made.

The *s command* is another two-letter command. The second character you type (the first character after the s) is a search character. The s command moves the cursor to the next occurence of the search character which appears to the right of the cursor. The cursor moves to the end of the current line if the search character does not occur to the right of the cursor. After you hit the s and before you hit the search character the prompt line will show "edit: search".

The *u command* moves the cursor up rapidly. The cursor keeps moving until it reaches the first line of the file or until you hit any key.

The *x command* is another two-letter command. The second character you type replaces the character under the cursor. The prompt line will show "edit: eXchange" until you hit the second character. If you hit a control character no change is made and the x command is cancelled.

Insert Mode

Use insert mode to insert many lines of text at once. In insert mode anything

you type (except special keys) will be inserted into the text to the left of the cursor. You may use special keys in insert mode exactly as in edit mode.

Command Mode

You use command mode to do potentially dangerous things like making multiple changes to the buffer or like updating your files. For this reason, all command-mode commands require you to type several characters with a carriage return at the end.

The editor starts off in command mode after you invoke it from your operating system.

You may use either the edit key or the insert key to exit from command mode. You can also use the g command-mode command or the find command-mode command to leave command mode. When the editor leaves command mode the screen is redrawn to show the current line.

The cursor is on the bottom line of the screen whenever the editor is in command mode. This line is called the command line. As you type commands, what you type will be shown on this line. Use either the delete character key or the right key to delete the last character you typed. Use the undo key to delete the entire command line.

Several command-mode commands take arguments. There are three types of arguments: range arguments, file names and line numbers. In the description below, range arguments will be denoted by <line range>, file name arguments will be denoted by <filename> and line numbers will be denoted by <n>.

Range arguments denote the range of lines in the buffer for which a command will take effect. For example,

```
list 25 50
```

means list all lines from line 25 to line 50 inclusive,

```
list
```

means list all the lines of the buffer, and

```
list 300
```

means list line 300 and all following lines. Range arguments, if present, must be nonnegative integers. If two numbers are entered, the first must be no larger than the second. 0 is equivalent to 1. A number larger than the largest line number is equivalent to the largest line number.

Filename arguments are the name of a file to be used in a command. For example,

```
load myfile.doc
```

causes the file myfile.doc to replace whatever is in the buffer. Note that no quotation marks are used around filename arguments. Filename arguments optionally may be preceded by disk drive names. For example,

```
load b:myfile.doc
```

loads the buffer from the file myfile.doc which must be found on drive b.

The editor protects you from destroying the buffer unintentionally. If a command would cause the buffer to be erased and you have not already saved the buffer on a disk file the editor will ask:

```
buffer not saved. proceed ?
```

Command	Mnemonic	Function
space	(over)	move cursor right 1 character
b	(Begin)	move cursor to start of line
c	(Command)	enter command mode
d	(Down)	scroll screen down
e	(End)	move cursor to end of line
g<n>	(Go)	go to line number <n>
i	(Insert)	enter insert mode
k<char>	(Kill)	delete from cursor up to <char>
s<char>	(Search)	move cursor right to H char K
u	(Up)	scroll screen up
x<char>	(eXchange)	replace cursor with <char>

Figure 4: Edit Mode Commands
Edit mode commands are one letter long. They may be typed in either upper or lower case.

If you type y or Y, the command will be done and the buffer will be erased. If you type anything else no change will be made to the buffer and the message "cancelled" will be printed on the screen. Note, however, that the delete command does *not* protect you in this way. Use the delete command with caution.

You may interrupt or cancel the change, find, list and search commands. If you type any character *except* a blank the command will be cancelled immediately. If you type a blank the command will be temporarily suspended. Type another blank to continue, or any other character to cancel the command.

The following paragraphs discuss all the command-mode commands. Figure 5 gives a short summary of how to invoke these commands and how to use them.

append <filename>

The append command inserts the named file into the buffer after the current line. In other words, the position of the cursor affects where the file will be inserted. This command adds to the buffer; it does not delete what is already there.

change <line range>

The change command searches all lines in <line range> for an instance of a search mask. If the search mask is found it is replaced by the change mask. After you type change <line range> the command will ask:

```
search mask ?
```

At this point type the pattern you are searching for, followed by a carriage return. Then you will be asked:

```
change mask ?
```

Now you type the pattern you want the search mask to be replaced by. For example,

```
change 100 300
search mask ? abc
change mask ? xyz
```

will change abc to xyz in all lines from 100 to 300. Thus, the line:

```
know your abc's
```

would be changed into:

```
know your xyz's
```

Only the first instance of abc on each line would be changed. The line:

```
know your abc's. The abc's are
    very important.
```

would be changed to:

know your xyz's. The abc's are
very important.

Question marks may be used in both the search mask and the change mask. A question mark in the search mask matches any character at all. The search mask:

12?34

will match any string of five character which starts with 12 and ends with 34. A question mark in the change mask matches the letter that matched the corresponding question mark in the search mask. For example:

change 99 109
search mask ? trailing?
change mask ? ?leading

would change the line:

trailingA

into the line:

Aleading

The question mark in the search mask matched the "A", so the question mark in the change mask became an "A". You can use as many question marks as you like, but there must be at least as many question marks in the search mask as there are in the change mask. As another example:

change
search mask ? wh??e
change mask ? fath??

would change the line:

where is here.

to the line:

father is here.

If the first character of the search mask is an up arrow (↑), the search mask is anchored to the start of the line. In other words, the search mask will only match patterns which start at column 0. For example, the search mask:

↑abc

will only match lines that contain abc in columns 0, 1 and 2. Anchoring the search mask dramatically increases the speed of the change command.

An up arrow which is not the first character of the search mask is treated as a regular up arrow. An up arrow in the change mask never has any special significance.

clear

The clear command erases the entire buffer. If you have not saved the buffer you will be asked whether you want to proceed.

delete <line range>

The delete command erases all lines in the line range. Care must be used with this command because no check is made to see whether the buffer has been saved.

dos

The dos command exits the editor and returns to the operating system. If the buffer has not been saved, this command makes sure that you really wanted to exit without saving the buffer.

find

The find command searches for the next occurrence of a search mask. If the search mask is found, the editor switches to edit mode and the cursor is set to the start of the pattern. If the search mask is not found the editor remains in command mode.

You specify the search mask exactly as in the change command.

g <n>

The g command exits to edit mode. Line <n> is drawn on the top line of the window and becomes the current line. Typing g without the <n> is the same as typing the edit key.

list <line range>

The list command lists all lines in the line range to the current list device, i.e., the printer. Lines are formatted just as they are on the screen, but the length of the print line, not the width of the screen, determines where lines will be truncated if they are too long. Listing may be stopped at any time by typing any key.

load <filename>

The load command replaces whatever is in the buffer by the contents of the named file. If the buffer has not been saved on disk when the load command is given, the editor makes sure you really wanted to erase the previous contents of the buffer. If the named file exists, then <filename> becomes "the current file" and <filename> is shown on the prompt line. This file becomes the file used in the save and resave commands. The current file name does not change if the named file does not exist.

After the file has been completely read from the disk into the buffer the screen is redrawn to show the first several lines of the file.

name <filename>

The name command sets the name of the file that will be used in the save and resave commands. This command is not often used because the load command is the usual way to set this file's name. The name command does not cause any action immediately; it simply affects which file the save and resave commands will use.

resave

The resave command saves the entire buffer in the current file. The name of the current file is set by either the load or name commands and always appears on the prompt line. Note that this command does *not* take any arguments. All arguments to this command are ignored.

The resave command requires that the current file already exist; if the file does not exist a warning is printed and nothing happens.

save

The save command works just like the resave command except that the file must *not* exist for the command to work. If the file does exist a warning is printed and nothing happens. This protection ensures that files cannot be overwritten inadvertently.

search <line range>

The search command prints on the screen all lines in <line range> which contain an instance of the search mask. As soon as the search command is typed the editor will ask:

search mask ?

Now type the search mask. Just as in the change command, a question mark matches any character and a leading up arrow anchors the search mask to the start of the line.

tabs <n>

The tabs command controls how lines are listed on the screen and on the printer. Specifically, the tabs command sets tab stops every <n> columns. <n> must be an integer. The command tabs 0 is equivalent to the command tabs 1.

Bringing Up & Changing the Editor

Figure 6 shows what equipment you need to have in order to use the editor without making *any* changes to the program. The code in the listings should work on any 8080, Z80, or 8085 machine that runs the CP/M operating system. You will also need quite a bit of memory. The editor itself uses about 23K bytes and the file you are editing must fit in the memory that is left. You also need a display screen that can be made to directly address the cursor. By that I mean that there must be some way to move the cursor wherever you like in one step. Every video board and terminal that I am aware of has this capability.

If you don't have access to the CP/M operating system then you will certainly have to make at least a few changes to the editor in order to get it up and running. But don't despair quite yet; you should only have to change the programs in the file ed8.c. Basically, these routines mask the idiosyncrasies of your operating system from the rest of the editor. If you find that you must change a routine not on this file in order to get the editor running on some other system, I'd like to hear about it. But I don't think you will.

Those of you with the CP/M 1.4 operating system will have to make only a very minor change to the routines syscstat(), syscin() and syscout() on file ed8.c. The listings contain the CP/M 1.4 versions of these routines commented out. Just comment out the CP/M 2.2 versions and remove the comments from around the CP/M 1.4 versions.

Those of you with displays that *only* work in block mode are going to have to work pretty hard to use this editor. (If you don't know what a block mode terminal is, rejoice. You don't have one of the beasts.) A somewhat uneducated guess is that you may have to rewrite all the routines in file ed6.ccc and many of the routines in file ed2.c. Frankly, I don't think it is worth the trouble. If somebody out there does get the editor working with a block-mode terminal I'd like to hear about *that*, too.

The files ed1.ccc and ed6.ccc contain code which is produced by the configuration program. These files tailor the editor to a particular display device. If for some reason you do not want to use the configuration program, you will have to create your own versions of these files.

One of the things that I learned while doing this project is that it is possible to change drastically how the editor looks to the user without really changing the guts the editor at all. From a programming point of view, the choice of which commands to include and which to exclude is pretty much arbitrary. Barry Dobyns' article about the MINCE editor in the April 1981 issue of *Dr. Dobb's Journal* is a good source of ideas about what you might want to add to this editor.

Everyone will have a favorite command or two that has been left out of this editor. I miss being able to move blocks of text from one area of the file to another. Another improvement might be to have a new line generated automatically in insert mode if the cursor reaches the end of the line. This is essentially an "auto-split" feature. These changes should be easy to make if you are so inclined.

The present version of this program limits editing to those files that can fit entirely into memory. This is not a fundamental problem with the editor. In fact, the first thing I designed about the editor was a scheme to edit arbitrarily long files. Essentially, what one has to do is to rewrite about half of the buffer module (file ed10.c). If there is enough interest, I may write another article telling how to do this.

Command	Arguments	Function
append	<filename>	insert file into buffer at cursor
change	<line range>	make changes to lines in <line range>
clear		erase the buffer
delete	<line range>	deletes lines in <line range>
dos		exit from editor to operating system
find		search for a pattern; enter edit mode
g	<n>	enter edit mode and go to line <n>
list	<line range>	print lines in <line range> to printer
load	<filename>	replace buffer with file
name	<filename>	set name of current file
save		save buffer to a new file
search	<line range>	print all lines that match a pattern
resave		save buffer to existing file
tabs	<n>	set tab stops at every <n> columns

Figure 5: Command Mode Commands
All command mode commands end with a carriage return. They may be typed in either upper or lower case.

How to Get This Editor

This editor is in the public domain. That means you can use it in any way you wish. You can move the editor to other machines, copy it however you like, and make whatever changes you think would make it better. In fact, I strongly encourage you to do these things.

If you have the CP/M operating system, you can get this editor and several other programs from me. They come on two 8-inch, soft-sectored, single-density, IBM-compatible diskettes. On these diskettes are the following:

1. Source code for the editor.
2. Source code for the configuration program.
3. Source code for the Small-C compiler as distributed by The Code Works (TM).
4. Source code for a text formatter adapted from the FORMAT program in the book, *Software Tools*, by Kernighan and Plauger, published by Addison-Wesley.
5. Complete documentation.

The cost for all the above programs is $50. I am making the cost somewhat higher than the cost of just copying the disks because I would prefer to send out just a few copies. It's fine with me if you get your copy from somebody who already has one.

I cannot offer to provide extensive support for these programs. That might quickly take all of my time. I will, however, pay one dollar to anyone who finds a bug in this program. Sorry, that's the best I can do.

If you use these programs and find them useful I hope you will add something of your own to them. I think this is a good editor, but I'm sure some of you will think of improvements. I'd like to

read about your ideas in *Dr. Dobb's Journal.*

Technical Details

Now that we have seen what the editor does, you might be interested in the technical details of how the editor does what it does. You can skip this section if these details are of no use to you. But please, if you do intend to make changes to this editor, make sure you understand the way the editor is constructed. If you don't, you are going to do violence to the design behind the editor.

The screen editor is organized into separate modules, also known as abstract data types. If you understand how and why the program is organized into these modules then you understand the basics of the program. I followed three rules religiously while writing this program. These rules tell how modules are designed. If you modify this program I want you to follow them too. Here they are:

Rule 1:

No program may use any data contained in another module. That is, all access to data contained in a module must be via the "access routines" of that module.

Rule 2:

No access routine may reveal to the outside world the internal representation of data inside the access routine's own module. In other words, the access routines of a module only indicate what the module does, not how the module does it.

Rule 3:

Every significant (i.e., relatively complex) data structure resides in one and only one module.

Each module, then, is responsible for hiding the details about some important data structure. To be more precise, each module hides how the module's internal data is represented. The idea is that the organization of the data may be changed inside a module without the programs that use the module knowing that anything has changed.

Modules not only contain data; just as importantly they also contain access routines by which procedures outside the module may use or change the data. The access routines are to be considered part of the module's data structure. I will sometimes speak of the "associated operations" of a module or data structure. By that I will mean the externally visible access routines of the module.

Since modules (or data structures) have associated operations, the concept of data structure widens to include dynamic devices such as disk drives, video screens, and operating systems! Indeed, I use modules to shield the rest of the program from the details of such devices.

Rule 1 says that programs outside a module can only use or change the data inside the module by calling the "access routines" of the module. No messing with other module's data is allowed.

Rule 2 says that these access routines are functions of the data. That is, the values returned and the effects produced by the access routines are independent of how the data of the module is represented. Rule 2 eliminates data dependencies between modules.

Rule 2 doesn't quite say enough. Not only must access routines hide their modules representation of data, access routines must provide a functional view of the module to the outside world. In other words, there must not be complex dependencies between access functions in the same modules. (By definition there are not dependencies between access functions of different modules.) Access functions can be linked in subtle ways. For example, it might be that the access functions of a module must be called in only certain orders. Such dependencies should be avoided if at all possible.

Rule 3 says that all important data structures are protected from meddling from outside sources, or conversely, no program needs to know (or can know) about details of data structures outside it's own module.

A corollary of these rules is that there are no important global data structures. Indeed, there is seldom, if ever, a need for any global variables at all. Variables may be shared by more than one routine, but all such routines will reside in the same module.

Not every routine needs to be a member of a module. Those routines that are not a member of any module know the details of none of the important data structures of the program.

There is a certain optimal complexity for the data structure enclosed by a module. If the structure is too complex then too many routines (all of them inside the module!) know details of the data structure. On the other hand, if a

(Continued on page 34)

Computer: 8080, 8085, or Z80 (4 mhz or faster preferred)

Operating system: CP/M

Main Memory: at least 32K (editor uses 23K)

Display: video screen with direct cursor addressing

Figure 6: What You Need to Use the Editor
You will have to make changes to the editor if you don't have the kind of system listed above.

Name of Module	File Name
window module	ed4.c
buffer module	ed10.c
line format module	ed5.c
prompt line module	ed7.c
terminal output module	ed6.ccc
operating system module	ed8.c
command module	ed2.c

Figure 7: Modules
The editor is made up of 7 modules. Each module controls an important data structure. Each module is on a separate file; modules do not share data.

Screen Oriented Editor

See bug fixes, DDJ Bound Volume page 193.

```
b:ed.c                  May 11, 1981                    page 1

/* Include file for screen editor
 *
 * Source: ed.c
 * Version: Feb. 10, 1980.
 */

#include ed0.c          /* constant (system) globals */
#include ed1.ccc        /* user defined globals */
#include ed2.c          /* main program */
#include ed3.c          /* command mode commands */
#include ed4.c          /* edit mode module */
#include ed5.c          /* output format module */
#include ed6.ccc        /* display screen module */
#include ed7.c          /* prompt line module */
#include ed8.c          /* operating system module */
#include ed9.c          /* utility routines */
#include ed10.c         /* memory buffer module */
#include c80lib         /* machine language library */
```

```
   ed0.c                September 8, 1981              page 1

/* Screen editor:  non-user defined globals
 *
 * Source: ed0.c
 * Version: June 19, 1980.
 */


/* Define global constants */

/* Define constants describing a text line */

#define MAXLEN  133     /* max chars per line */
#define MAXLEN1 134     /* MAXLEN + 1 */

/* Define operating system constants */

#define SYSFNMAX 15     /* CP/M file name length + 1 */

/* Define misc. constants */

#define EOS     0       /* code sometimes assumes \0 */
#define OK      1
#define ERR     -1      /* error.  must be <0 */
#define EOF     -2      /* end of file.  must be <0 */
#define YES     1       /* must be nonzero */
#define NO      0
#define CR      13      /* carriage return */
#define LF      10      /* line feed */
#define TAB     9       /* tab character */
#define HUGE    32000   /* practical infinity */
```

```
ed1.ccc                 September 8, 1981               page 1


/*
Screen editor:  special key definitions
Source:  ed1.ccc
This file was created by the configuration program:
Version 2:  September 6, 1981.
*/

/*
Define which keys are used for special edit functions.
*/

#define UP1 10
#define DOWN1 13
#define UP2 21
#define DOWN2 4
#define LEFT1 8
#define RIGHT1 18
#define INS1 14
#define EDIT1 5
#define ESC1 27
#define DEL1 127
#define ZAP1 26
#define ABT1 24
#define SPLT1 19
#define JOIN1 16

/*
Define length and width of screen and printer.
*/
#define SCRNW 64
#define SCRNW1 63
#define SCRNL 16
#define SCRNL1 15
#define SCRNL2 14
#define LISTW 132
```

```
ed2.c                   September 8, 1981               page 1

/* Screen editor:  main program
 *
 * Source:  ed2.c
 * Version: September 5, 1981.
 */

/* define signon message */

#define SIGNON "small-c editor, version 2:  September 5, 1981."

/* the main program dispatches the routines that
 * handle the various modes.
 */

#define CMNDMODE 1      /* enter command mode flag */
#define INSMODE  2      /* enter insert modes flag */
#define EDITMODE 3      /* enter edit mode flag */
#define EXITMODE 4      /* exit editor flag */

main()
{
int mode;
        /* fmt output by default goes to screen */
        fmtassn(NO);
        /* set tabs, clear the screen and sign on */
        fmtset(8);
        outclr();
        outxy(0,SCRNL1);
        message(SIGNON);
        outxy(0,1);
        /* clear filename[] for save(), resave() */
        name("");
        /* clear the main buffer */
        bufnew();
        /* start off in command mode */
        mode=CMNDMODE;
        /* get null line 1 for edit() */
        edgetln();
        while(1){
                if (mode==EXITMODE) {
                        break;
                }
                else if (mode==CMNDMODE) {
                        mode=command();
                }
                else if (mode==EDITMODE) {
                        mode=edit();
                }
                else if (mode==INSMODE) {
                        mode=insert();
                }
                else {
                        syserr("main: no mode");
                        mode=EDITMODE;
                }
        }
}

/*
 * handle edit mode.
 * dispatch the proper routine based on one-character commands.
 */
```

```
ed2.c                   September 8, 1981               page 2


edit()
{
char buffer[SCRNW1];
int v;
int x,y;
char c;
        /* we can't do edgetln() or edgo() here because
         * those calls reset the cursor.
         */
        pmtedit();
        while(1){
                /* get command */
                c=tolower(syscin());
                if ((c==ESC1)|(c=='c')) {
                        /* enter command mode. */
                        return(CMNDMODE);
                }
```

```
        else if ((c==INS1)|(c=='i')) {
                /* enter insert mode */
                return(INSMODE);
        }
        else if (special(c) == YES) {
                if ((c == UP1)|(c == DOWN1)) {
                        return(INSMODE);
                }
                else {
                        continue;
                }
        }
        else if (control(c)==YES) {
                continue;
        }
        else if (c==' ') {
                edright();
                pmtcol();
        }
        else if (c=='b') {
                edbegin();
                pmtcol();
        }
        else if (c=='d') {
                /* scroll down */
                pmtmode("edit: scroll");
                while (bufnrbot()==NO) {
                        if (chkkey()==YES) {
                                break;
                        }
                        if (eddn()==ERR) {
                                break;
                        }
                }
                pmtedit();
        }
        else if (c=='e') {
                edend();
                pmtcol();
        }
        else if (c=='g') {
                /* save x,y in case don't get number */
                x=outgetx();
                y=outgety();

ed2.c                September 8, 1981                page 3

                pmtcmnd("edit: goto: ",buffer);
                if(number(buffer,&v)) {
                        edgo(v,0);
                }
                else {
                        outxy(x,y);
                }
                pmtedit();
        }
        else if (c=='k') {
                pmtmode("edit: kill");
                c=syscin();
                if ((special(c)==NO) &
                    (control(c)==NO)) {
                        edkill(c);
                }
                pmtedit();
        }
        else if (c=='s') {
                pmtmode("edit: search");
                c=syscin();
                if ((special(c)==NO) &
                    (control(c)==NO)) {
                        edsrch(c);
                }
                pmtedit();
        }
        else if (c=='u') {
                /* scroll up */
                pmtmode("edit: scroll");
                while (bufattop()==NO) {
                        if (chkkey()==YES) {
                                break;
                        }
                        if (edup()==ERR) {
                                break;
                        }
                }
                pmtedit();
        }
        else if (c=='x') {
                pmtmode("edit: eXchange");
                c=syscin();
                if ((special(c)==NO) &
                    (control(c)==NO)) {
                        edchng(c);
                }
                pmtedit();
        }
        /* do nothing if command not found */
    }
}

/* insert mode.
 * in this mode the UP1, UP2 keys reverse their roles,
 * as do the DOWN1, and DOWN2 keys.
 */

insert()
{
char c;
    pmtmode("insert");

ed2.c                September 8, 1981                page 4

    while (1) {
        /* get command */
        c=syscin();
        if (c==ESC1) {
            /* enter command mode */
            return(CMNDMODE);
        }
        else if (c==EDIT1) {
            /* enter edit mode */
            return(EDITMODE);
        }
        else if (c==INS1) {
            /* do nothing */
            ;
        }
        else if (special(c)==YES) {
            if ((c == UP2)|(c == DOWN2)) {
                return(EDITMODE);
            }
            else {
                continue;
            }
        }
        else if (control(c)==YES) {
            /* ignore non-special control chars */
            continue;
        }
        else {
            /* insert one char in line */
            edins(c);
            pmtcol();
        }
    }
}

/* return YES if c is a control char */

control(c) char c;
{
    if (c==TAB) {
        return(NO);     /* tab is regular */
    }
    else if (c>=127) {
        return(YES);    /* del or high bit on */
    }
    else if (c<32) {
        return(YES);    /* control char */
    }
    else {
        return(NO);     /* normal */
    }
}

/*
 * handle the default actions of all special keys.
 * return YES if c is one of the keys.
 */

special(c) char c;
{
int k;
    if (c==JOIN1) {

ed2.c                September 8, 1981                page 5

        edjoin();
        pmtline();
        return(YES);
    }
    if (c==SPLT1) {
        edsplit();
        pmtline();
        return(YES);
```

Screen Oriented Editor

(Listing continued, text begins on page 18)

```
        }
        if (c==ABT1) {
                edabt();
                pmtcol();
                return(YES);
        }
        else if (c==DEL1) {
                eddel();
                pmtcol();
                return(YES);
        }
        else if (c==ZAP1) {
                edzap();
                pmtline();
                return(YES);
        }
        else if (c==UP2) {
                /* move up */
                edup();
                pmtline();
                return(YES);
        }
        else if (c==UP1) {
                /* insert up */
                ednewup();
                pmtline();
                return(YES);
        }
        else if (c==DOWN2) {
                /* move down */
                eddn();
                pmtline();
                return(YES);
        }
        else if (c==DOWN1) {
                /* insert down */
                ednewdn();
                pmtline();
                return(YES);
        }
        else if (c==LEFT1) {
                edleft();
                pmtcol();
                return(YES);
        }
        else if (c==RIGHT1) {
                edright();
                pmtcol();
                return(YES);
        }
        else {
                return(NO);
        }
}
```

ed2.c September 8, 1981 page 6

```
/*
 * command() dispatches command routines while
 * in command mode.
 */

command()
{
int v;
char c;
char args[SCRNW1];
char *argp;
int topline;
int ypos;
int oldline;
int k;
        /* command mode commands may move the current line.
         * command mode must save the current line on entry
         * and restore it on exit.
         */
        edrepl();
        /* remember how the screen was drawn on entry */
        oldline=bufln();
        ypos=outgety();
        topline=oldline-ypos+1;
        while(1) {
                outxy(0,SCRNL1);
                fmtcrlf();
                pmtmode("command:");
                getcmnd(args,0);
                fmtcrlf();
                pmtline();
                c=args[0];
                if ((c==EDIT1)|(c==INS1)) {
                        /* redraw screen */
                        if (oldline==bufln()) {
                                /* get current line */
                                edgetln();
                                /* redraw old screen */
                                bufout(topline,1,SCRNL1);
                                outxy(0,ypos);
                        }
                        else {
                                /* update line and screen */
                                edgo(bufln(),0);
                        }
                        if (c==EDIT1) {
                                return (EDITMODE);
                        }
                        else {
                                return (INSMODE);
                        }
                }
                else if (tolower(args[0])=='g'){
                        argp=skipbl(args+1);
                        if (argp[0]==EOS) {
                                edgo(oldline,0);
                                return(EDITMODE);
                        }
                        else if (number(argp,&v)==YES) {
                                edgo(v,0);
                                return(EDITMODE);
```

ed2.c September 8, 1981 page 7

```
                        }
                        else {
                                message("bad line number");
                        }
                }
                else if (lookup(args,"append")) {
                        append(args);
                }
                else if (lookup(args,"change")) {
                        change(args);
                }
                else if (lookup(args,"clear")) {
                        clear();
                }
                else if (lookup(args,"delete")) {
                        delete(args);
                }
                else if (lookup(args,"dos")) {
                        if (chkbuf()==YES) {
                                return (EXITMODE);
                        }
                }
                else if (lookup(args,"find")) {
                        if ((k = find()) >= 0) {
                                edgo(bufln(),k);
                                return(EDITMODE);
                        }
                        else {
                                /* get current line */
                                bufgo(oldline);
                                edgetln();
                                /* stay in command mode */
                                message("pattern not found");
                        }
                }
                else if (lookup(args,"list")) {
                        list(args);
                }
                else if (lookup(args,"load")) {
                        load(args);
                }
                else if (lookup(args,"name")) {
                        name(args);
                }
                else if (lookup(args,"resave")) {
                        resave();
                }
                else if (lookup(args,"save")) {
                        save();
                }
                else if (lookup(args,"search")) {
                        search(args);
                }
                else if (lookup(args,"tabs")) {
                        tabs(args);
                }
                else {
                        message("command not found");
                }
```

```
        }
}
```

ed2.c September 8, 1981 page 8

```
/* return YES if line starts with command */

lookup(line,command) char *line, *command;
{
        while(*command) {
                if (tolower(*line++)!=*command++) {
                        return(NO);
                }
        }
        if((*line==EOS)|(*line==' ')|(*line==TAB)) {
                return(YES);
        }
        else {
                return(NO);
        }
}

/* get next command into argument buffer */

getcmnd(args,offset) char *args; int offset;
{
int j,k;
char c;
        outxy(offset,outgety());
        outdeol();
        k=0;
        while ((c=syscin())!=CR) {
                if ((c==EDIT1)|(c==INS1)) {
                        args[0]=c;
                        return;
                }
                if ((c==DEL1)|(c==LEFT1)) {
                        if (k>0) {
                                outxy(offset,outgety());
                                outdeol();
                                k--;
                                j=0;
                                while (j<k) {
                                        outchar(args[j++]);
                                }
                        }
                }
                else if (c==ABT1) {
                        outxy(offset,outgety());
                        outdeol();
                        k=0;
                }
                else if ((c!=TAB)&((c<32)|(c==127))) {
                        /* do nothing */
                        continue;
                }
                else {
                        if ((k+offset)<SCRNW1) {
                                args[k++]=c;
                                outchar(c);
                        }
                }
        }
        args[k]=EOS;
}
```

ed3.c September 8, 1981 page 1

```
/*
 * Screen editor:  command mode commands
 *
 * Source: ed3.c
 * Version: September 5, 1981.
 *
 */

/* data global to these routines */

char filename[SYSFNMAX];


/* append command.
 * load a file into main buffer at current location.
 * this command does NOT change the current file name.
 */

append(args) char *args;
{
char buffer[MAXLEN];            /* disk line buffer */
int file;
int n;
int topline;
char locfn[SYSFNMAX];           /* local file name */
        /* get file name which follows command */
        if (name1(args,locfn)==ERR) {
                return;
        }
        if (locfn[0]==EOS) {
                message("no file argument");
                return;
        }
        /* open the new file */
        if ((file=sysopen(locfn,"r"))==ERR) {
                message("file not found");
                return;
        }
        /* read the file into the buffer */
        while ((n=readline(file,buffer,MAXLEN))>=0) {
                if (n>MAXLEN) {
                        message("line truncated");
                        n=MAXLEN;
                }
                if (bufins(buffer,n)==ERR) {
                        break;
                }
                if (bufdn()==ERR) {
                        break;
                }
        }
        /* close the file */
        sysclose(file);
        /* redraw the screen so topline will be at top
         * of the screen after command() does a CR/LF.
         */
        topline=max(1,bufln()-SCRNL2);
        bufout(topline,2,SCRNL2);
        bufgo(topline);
}

/* global change command */
```

ed3.c September 8, 1981 page 2

```
change(args) char *args;
{
char oldline[MAXLEN1];          /* reserve space for EOS */
char newline[MAXLEN1];
char oldpat[MAXLEN1];
char newpat[MAXLEN1];
int from, to, col, n, k;
        if (get2args(args,&from,&to)==ERR) {
                return;
        }
        /* get search and change masks into oldpat, newpat */
        fmtsout("search mask ?  ",0);
        getcmnd(oldpat,15);
        fmtcrlf();
        if (oldpat[0]==EOS) {
                return;
        }
        pmtline();
        fmtsout("change mask ?  ",0);
        getcmnd(newpat,15);
        fmtcrlf();
        /* make substitution for lines between from, to */
        while (from<=to) {
                if (chkkey()==YES) {
                        break;
                }
                if (bufgo(from++)==ERR) {
                        break;
                }
                if (bufatbot()==YES) {
                        break;
                }
                n=bufgetln(oldline,MAXLEN);
                n=min(n,MAXLEN);
                oldline[n]=EOS;
                /* '^' anchors search */
                if (oldpat[0]=='^') {
                        if (amatch(oldline,oldpat+1,0)==YES) {
                                k=replace(oldline,newline,
                                        oldpat+1,newpat,0);
                                if (k==ERR) {
                                        return;
                                }
                                fmtcrlf();
                                putdec(bufln(),5);
                                fmtsout(newline,5);
                                outdeol();
                                bufrepl(newline,k);
                        }
```

(Continued on next page)

Screen Oriented Editor

(Listing continued, text begins on page 18)

```
                continue;
        }
        /* search oldline for oldpat */
        col=0;
        while (col<n) {
                if (amatch(oldline,oldpat,col++)==YES){
                        k=replace(oldline,newline,
                                oldpat,newpat,col-1);
                        if (k==ERR) {
                                return;
                        }
                        fmtcrlf();
```



```
                        putdec(bufln(),5);
                        fmtsout(newline,5);
                        outdeol();
                        bufrepl(newline,k);
                        break;
                }
        }
    }
    fmtcrlf();
}

/* clear main buffer and file name */

clear()
{
    /* make sure it is ok to clear buffer */
    if (chkbuf()==YES) {
        filename[0]=0;
        pmtfile("");
        outclr();
        outxy(0,SCRNL1);
        bufnew();
        message("buffer cleared");
    }
}

/* multiple line delete command */

delete(args) char *args;
{
int from, to;
    if (get2args(args,&from,&to)==ERR) {
        return;
    }
    if (from>to) {
        return;
    }
    /* go to first line to be deleted */
    if (bufgo(from)==ERR) {
        return;
    }
    /* delete all line between from and to */
    if (bufdeln(to-from+1)==ERR) {
        return;
    }
    /* redraw the screen */
    bufout(bufln(),1,SCRNL1);
}

/* search all lines below the current line for a pattern
 * return -1 if pattern not found.
 * otherwise, return column number of start of pattern.
 */

find()
{
    return(search1(bufln()+1,HUGE,YES));
}

/* list lines to list device */

list(args) char *args;
```



```
{
char linebuf[MAXLEN1];
int n;
int from, to, line, oldline;
    /* save the buffer's current line */
    oldline=bufln();
    /* get starting, ending lines to print */
    if (get2args(args,&from,&to)==ERR) {
        return;
    }
    /* print lines one at a time to list device */
    line=from;
    while (line<=to) {
        /* make sure prompt goes to console */
        fmtassn(NO);
        /* check for interrupt */
        if (chkkey()==YES) {
            break;
        }
        /* print line to list device */
        fmtassn(YES);
        if (bufgo(line++)!=OK) {
            break;
        }
        if (bufatbot()) {
            break;
        }
        n=bufgetln(linebuf,MAXLEN1);
        n=min(n,MAXLEN);
        linebuf[n]=CR;
        fmtsout(linebuf,0);
        fmtcrlf();
    }
    /* redirect output to console */
    fmtassn(NO);
    /* restore cursor */
    bufgo(oldline);
}

/* load file into buffer */

load (args) char *args;
{
char buffer[MAXLEN];    /* disk line buffer */
char locfn [SYSFNMAX];  /* file name until we check it */
int n;
int file;
int topline;
    /* get filename following command */
    if (name1(args,locfn)==ERR) {
        return;
    }
    if (locfn[0]==EOS) {
        message("no file argument");
        return;
    }
    /* give user a chance to save the buffer */
    if (chkbuf()==NO) {
        return;
    }
    /* open the new file */
    if ((file=sysopen(locfn,"r"))==ERR) {
```



```
        message("file not found");
        return;
    }
    /* update file name */
    syscopfn(locfn, filename);
    pmtfile(filename);
    /* clear the buffer */
    bufnew();
    /* read the file into the buffer */
    while ((n=readline(file,buffer,MAXLEN))>=0) {
        if (n>MAXLEN) {
            message("line truncated");
            n=MAXLEN;
        }
        if (bufins(buffer,n)==ERR) {
            break;
        }
        if (bufdn()==ERR) {
            break;
        }
    }
    /* close the file */
    sysclose(file);
    /* indicate that the buffer is fresh */
    bufsaved();
    /* set current line to line 1 */
    bufgo(1);
    /* redraw the screen so that topline will be
     * on line 1 after command() does a CR/LF.
     */
    topline=max(1,bufln()-SCRNL2);
    bufout(topline,2,SCRNL2);
    bufgo(topline);
}

/* change current file name */

name(args) char *args;
```

```
{
        name1(args,filename);
        pmtfile(filename);
}

/* check syntax of args.
 * copy to filename.
 * return OK if the name is valid.
 */

name1(args,filename) char *args, *filename;
{
        /* skip command */
        args=skiparg(args);
        args=skipbl(args);
        /* check file name syntax */
        if (syschkfn(args)==ERR) {
                return(ERR);
        }
        /* copy filename */
        syscopfn(args,filename);
        return(OK);
}
```

ed3.c September 8, 1981 page 6

```
/* save the buffer in an already existing file */

resave()
{
char linebuf[MAXLEN];
int file, n, oldline;
        /* make sure file has a name */
        if (filename[0]==EOS) {
                message("file not named");
                return;
        }
        /* the file must exist for resave */
        if ((file=sysopen(filename,"r"))==ERR) {
                message("file not found");
                return;
        }
        if (sysclose(file)==ERR) {
                return;
        }
        /* open the file for writing */
        if ((file=sysopen(filename,"w"))==ERR) {
                return;
        }
        /* save the current position of file */
        oldline=bufln();
        /* write out the whole file */
        if (bufgo(1)==ERR) {
                sysclose(file);
                return;
        }
        while (bufatbot()==NO) {
                n=bufgetln(linebuf,MAXLEN);
                n=min(n,MAXLEN);
                if (pushline(file,linebuf,n)==ERR) {
                        break;
                }
                if (bufdn()==ERR) {
                        break;
                }
        }
        /* indicate if all buffer was saved */
        if (bufatbot()){
                bufsaved();
        }
        /* close file and restore line number */
        sysclose(file);
        bufgo(oldline);
}

/* save the buffer in a new file */

save()
{
char linebuf[MAXLEN];
int file, n, oldline;
        /* make sure the file is named */
        if (filename[0]==EOS) {
                message("file not named");
                return;
        }
        /* file must NOT exist for save */
        if ((file=sysopen(filename,"r"))!=ERR) {
```

ed3.c September 8, 1981 page 7

```
                sysclose(file);
                message("file exists");
                return;
        }
        /* open file for writing */
        if ((file=sysopen(filename,"w"))==ERR) {
                return;
        }
        /* remember current line */
        oldline=bufln();
        /* write entire buffer to file */
        if (bufgo(1)==ERR) {
                sysclose(file);
                return;
        }
        while (bufatbot()==NO) {
                n=bufgetln(linebuf,MAXLEN);
                n=min(n,MAXLEN);
                if (pushline(file,linebuf,n)==ERR) {
                        break;
                }
                if (bufdn()==ERR) {
                        break;
                }
        }
        /* indicate buffer saved if good write */
        if (bufatbot()) {
                bufsaved();
        }
        /* restore line and close file */
        bufgo(oldline);
        sysclose(file);
}

/* global search command */

search(args) char *args;
{
int from, to;

        if (get2args(args,&from,&to)==ERR) {
                return;
        }
        search1(from, to, NO);
}

/* search lines for a pattern.
 * if flag == YES: stop at the first match.
 *                 return -1 if no match.
 *                 otherwise return column number of match
 * if flag == NO:  print all matches found.
 */

search1(from, to, flag) int from, to, flag;
{
char pat  [MAXLEN1];            /* reserve space for EOS */
char line [MAXLEN1];
int col, n;
        /* get search mask into pat */
        fmtsout("search mask ?  ",0);
        getcmnd(pat,15);
        fmtcrlf();
```

ed3.c September 8, 1981 page 8

```
        if (pat[0]==EOS) {
                return;
        }
        /* search all lines between from and to for pat */
        while (from<=to) {
                if (chkkey()==YES) {
                        break;
                }
                if (bufgo(from++)==ERR) {
                        break;
                }
                if (bufatbot()==YES) {
                        break;
                }
                n=bufgetln(line,MAXLEN);
                n=min(n,MAXLEN);
                line[n]=EOS;
                /* ^ anchors search */
                if (pat[0]=='^') {
                        if (amatch(line,pat+1,0)==YES) {
                                if (flag==NO) {
                                        fmtcrlf();
                                        putdec(bufln(),5);
                                        fmtsout(line,5);
                                        outdeol();
                                }
                                else {
                                        return(0);
```

(Continued on next page)

Screen Oriented Editor

(Listing continued, text begins on page 18)

```
                        }
                }
                continue;
        }
        /* search whole line for match */
        col=0;
        while (col<n) {
                if (amatch(line,pat,col++)==YES) {
                        if (flag==NO) {
                                fmtcrlf();
                                putdec(bufln(),5);
                                fmtsout(line,5);
                                outdeol();
                                break;
                        }
                        else {
                                return(col-1);
                        }
                }
        }
        /* all searching is finished */
        if (flag==YES) {
                return(-1);
        }
        else {
                fmtcrlf();
        }
}

/* set tab stops for fmt routines */

tabs(args) char *args;
{
```



```
int n, junk;
        if (get2args(args,&n,&junk)==ERR) {
                return;
        }
        fmtset(n);
}

/* return YES if buffer may be drastically changed */

chkbuf()
{
        if (bufchng()==NO) {
                /* buffer not changed. no problem */
                return(YES);
        }
        fmtsout("buffer not saved. proceed ?  ",0);
        pmtline();
        if (tolower(syscout(syscin()))!='y') {
                fmtcrlf();
                message("cancelled");
                return(NO);
        }
        else {
                fmtcrlf();
                return(YES);
        }
}

/* print message from a command */

message(s) char *s;
{
        fmtsout(s,0);
        fmtcrlf();
}

/* get two arguments the argument line args.
 * no arguments imply 1 HUGE.
 * one argument implies both args the same.
 */

get2args(args,val1,val2) char *args; int *val1, *val2;
{
        /* skip over the command */
        args=skiparg(args);
        args=skipbl(args);
        if (*args==EOS) {
                *val1=1;
                *val2=HUGE;
                return(OK);
        }
        /* check first argument */
        if (number(args,val1)==NO) {
                message("bad argument");
                return(ERR);
        }
        /* skip over first argument */
        args=skiparg(args);
        args=skipbl(args);
        /* 1 arg: arg 2 is HUGE */
        if (*args==EOS) {
                *val2=HUGE;
```



```
                return(OK);
        }
        /* check second argument */
        if (number(args,val2)==NO) {
                message("bad argument");
                return(ERR);
        }
        else {
                return(OK);
        }
}

/* skip over all except EOS, and blanks */

skiparg(args) char *args;
{
        while ((*args!=EOS)&(*args!=' ')) {
                args++;
        }
        return(args);
}

/* skip over all blanks */

skipbl(args) char *args;
{
        while (*args==' ') {
                args++;
        }
        return(args);
}

/* return YES if the user has pressed any key.
 * blanks cause a transparent pause.
 */
chkkey()
{
int c;
        c=syscstat();
        if (c==0) {
                /* no character at keyboard */
                return(NO);
        }
        else if (c==' ') {
                /* pause.  another blank ends pause */
                pmtline();
                if (syscin()==' ') {
                        return(NO);
                }
        }
        /* we got a nonblank character */
        return(YES);
}

/* anchored search for pattern in text line at column col.
 * return YES if the pattern starts at col.
 */

amatch(line,pat,col) char *line, *pat; int col;
{
int k;
        k=0;
```



```
        while (pat[k]!=EOS) {
                if (pat[k]==line[col]) {
                        k++;
                        col++;
                }
                else if ((pat[k]=='?')&(line[col]!=EOS)) {
                        /* question mark matches any char */
                        k++;
                        col++;
                }
                else {
                        return(NO);
                }
        }
        /* the entire pattern matches */
        return(YES);
```

```
}

/* replace oldpat in oldline by newpat starting at col.
 * put result in newline.
 * return number of characters in newline.
 */

replace(oldline,newline,oldpat,newpat,col)
char *oldline, *newline, *oldpat, *newpat; int col;
{
int k;
char *tail, *pat;
        /* copy oldline preceding col to newline */
        k=0;
        while (k<col) {
                newline[k++]=*oldline++;
        }
        /* remember where end of oldpat in oldline is */
        tail=oldline;
        pat=oldpat;
        while (*pat++!=EOS) {
                tail++;
        }
        /* copy newpat to newline.
         * use oldline and oldpat to resolve question marks
         * in newpat.
         */
        while (*newpat!=EOS) {
                if (k>MAXLEN-1) {
                        message("new line too long");
                        return(ERR);
                }
                if (*newpat!='?') {
                        /* copy newpat to newline */
                        newline[k++]=*newpat++;
                        continue;
                }
                /* scan for '?' in oldpat */
                while (*oldpat!='?') {
                        if (*oldpat==EOS) {
                                message(
                                "too many ?'s in change mask"
                                );
                                return(ERR);
                        }
                        oldpat++;
```



```
                        oldline++;
                }
                /* copy char from oldline to newline */
                newline[k++]=*oldline++;
                oldpat++;
                newpat++;
        }
        /* copy oldline after oldpat to newline */
        while (*tail!=EOS) {
                if (k>=MAXLEN-1) {
                        message("new line too long");
                        return(ERR);
                }
                newline[k++]=*tail++;
        }
        newline[k]=EOS;
        return(k);
}
```

```
/*
 * Screen editor:  window module
 *
 * Source:  ed4.c
 * Version: August 20, 1981.
 */

/* data global to this module */

char    editbuf[MAXLEN];        /* the edit buffer */
int     editp;                  /* cursor: buffer index */
int     editpmax;               /* length of buffer */
int     edcflag;                /* buffer change flag */

/* abort any changes made to current line */

edabt()
{
        /* get unchanged line and reset cursor */
        edgetln();
        edredraw();
        edbegin();
        edcflag=NO;
}

/* put cursor at beginning of current line */

edbegin()
{
        editp=0;
        outxy(0,outgety());
}

/* change editbuf[editp] to c
 * don't make change if line would become to long
 */

edchng(c) char c;
{
char oldc;
int k;
        /* if at right margin then insert char */
        if (editp>=editpmax) {
                edins(c);
                return;
        }
        /* change char and print length of line */
        oldc=editbuf[editp];
        editbuf[editp]=c;
        fmtadj(editbuf,editp,editpmax);
        k=fmtlen(editbuf,editpmax);
        if (k>SCRNW1) {
                /* line would become too long */
                /* undo the change */
                editbuf[editp]=oldc;
                fmtadj(editbuf,editp,editpmax);
        }
        else {
                /* set change flag, redraw line */
                edcflag=YES;
                editp++;
                edredraw();
```



```
        }
}

/* delete the char to left of cursor if it exists */

eddel()
{
int k;
        /* just move left one column if past end of line */
        if (edxpos() < outgetx()) {
                outxy(outgetx()-1, outgety());
                return;
        }
        /* do nothing if cursor is at left margin */
        if (editp==0) {
                return;
        }
        edcflag=YES;
        /* compress buffer (delete char) */
        k=editp;
        while (k<editpmax) {
                editbuf[k-1]=editbuf[k];
                k++;
        }
        /* update pointers, redraw line */
        editp--;
        editpmax--;
        edredraw();
}

/* edit the next line.  do not go to end of buffer */

eddn()
{
int oldx;
        /* save visual position of cursor */
        oldx=outgetx();
        /* replace current edit line */
        if (edrepl()!=OK) {
                return(ERR);
        }
        /* do not go past last non-null line */
        if (bufnrbot()) {
                return(OK);
        }
        /* move down one line in buffer */
        if (bufdn()!=OK) {
```

(Continued on next page)

Screen Oriented Editor

(Listing continued, text begins on page 18)

```
                return(ERR);
        }
        edgetln();
        /* put cursor as close as possible on this
         * new line to where it was on the old line.
         */
        editp=edscan(oldx);
        /* update screen */
        if (edatbot()) {
                edsup(bufln()-SCRNL2);
                outxy(oldx, SCRNL1);
        }
        else {
                outxy(oldx, outgety()+1);
        }
```



```
        return(OK);
}

/* put cursor at the end of the current line */

edend()
{
        editp=editpmax;
        outxy(edxpos(),outgety());

        /* comment out ----- put cursor at end of screen
        outxy(SCRNW1, outgety());
        ----- end comment out */
}

/* start editing line n
 * redraw the screen with cursor at position p
 */

edgo(n, p) int n, p;
{
        /* replace current line */
        if (edrepl()==ERR) {
                return(ERR);
        }
        /* go to new line */
        if (bufgo(n)==ERR) {
                return(ERR);
        }
        /* prevent going past end of buffer */
        if (bufatbot()) {
                if (bufup()==ERR) {
                        return(ERR);
                }
        }
        /* redraw the screen */
        bufout(bufln(),1,SCRNL1);
        edgetln();
        editp=min(p, editpmax);
        outxy(edxpos(), 1);
        return(OK);
}

/* insert c into the buffer if possible */

edins(c) char c;
{
int k;

        /* do nothing if edit buffer is full */
        if (editpmax>=MAXLEN) {
                return;
        }
        /* fill out line if we are past its end */
        if ((editp == editpmax) & (edxpos() < outgetx())) {
                k = outgetx() - edxpos();
                editpmax = editpmax + k;
                while (k-- > 0) {
                        editbuf [editp++] = ' ';
                }
                editp = editpmax;
        }
```



```
        /* make room for inserted character */
        k=editpmax;
        while (k>editp) {
                editbuf[k]=editbuf[k-1];
                k--;
        }
        /* insert character. update pointers */
        editbuf[editp]=c;
        editp++;
        editpmax++;
        /* recalculate print length of line  */
        fmtadj(editbuf,editp-1,editpmax);
        k=fmtlen(editbuf,editpmax);
        if (k>SCRNW1) {
                /* line would become too long */
                /* delete what we just inserted */
                eddel();
        }
        else {
                /* set change flag, redraw line */
                edcflag=YES;
                edredraw();
        }
}

/* join (concatenate) the current line with the one above it */

edjoin()
{
int k;

        /* do nothing if at top of file */
        if (bufattop()) {
                return;
        }
        /* replace lower line temporarily */
        if (edrepl() != OK) {
                return;
        }
        /* get upper line into buffer */
        if (bufup() != OK) {
                return;
        }
        k = bufgetln(editbuf, MAXLEN);
        /* append lower line to buffer */
        if (bufdn() != OK) {
                return;
        }
        k = k + bufgetln(editbuf+k, MAXLEN-k);
        /* abort if the screen isn't wide enough */
        if (k>SCRNW1) {
                return;
        }
        /* replace upper line */
        if (bufup() != OK) {
                return;
        }
        editpmax = k;
        edcflag = YES;
        if (edrepl() != OK) {
                return;
        }
```



```
        /* delete the lower line */
        if (bufdn() != OK) {
                return;
        }
        if (bufdel() != OK) {
                return;
        }
        if (bufup() != OK) {
                return;
        }
        /* update the screen */
        if (edattop()) {
                edredraw();
        }
        else {
                k = outgety() - 1;
                bufout(bufln(),k,SCRNL-k);
                outxy(0,k);
                edredraw();
        }
}

/* delete chars until end of line or c found */

edkill(c) char c;
{
int k,p;
        /* do nothing if at right margin */
        if (editp==editpmax) {
                return;
        }
        edcflag=YES;
        /* count number of deleted chars */
        k=1;
```

```
        while ((editp+k)<editpmax) {
                if (editbuf[editp+k]==c) {
                        break;
                }
                else {
                        k++;
                }
        }
        /* compress buffer (delete chars) */
        p=editp+k;
        while (p<editpmax) {
                editbuf[p-k]=editbuf[p];
                p++;
        }
        /* update buffer size, redraw line */
        editpmax=editpmax-k;
        edredraw();
}

/* move cursor left one column.
 * never move the cursor off the current line.
 */

edleft()
{
int k;

        /* if past right margin, move left one column */
```



```
        if (edxpos() < outgetx()) {
                outxy(max(0, outgetx()-1), outgety());
        }
        /* inside the line.  move left one character */
        else if (editp!=0) {
                editp--;
                outxy(edxpos(),outgety());
        }
}

/* insert a new blank line below the current line */

ednewdn()
{
int k;
        /* make sure there is a current line and
         * put the current line back into the buffer.
         */
        if (bufatbot()) {
                if (bufins(editbuf,editpmax)!=OK) {
                        return;
                }
        }
        else if (edrepl()!=OK) {
                        return;
        }
        /* move past current line */
        if (bufdn()!=OK) {
                return;
        }
        /* insert place holder:  zero length line */
        if (bufins(editbuf,0)!=OK) {
                return;
        }
        /* start editing the zero length line */
        edgetln();
        /* update the screen */
        if (edatbot()) {
                /* note: bufln() >= SCRNL */
                edsup(bufln()-SCRNL2);
                outxy(edxpos(),SCRNL1);
        }
        else {
                k=outgety();
                bufout(bufln(),k+1,SCRNL1-k);
                outxy(edxpos(),k+1);
        }
}

/* insert a new blank line above the current line */

ednewup()
{
int k;
        /* put current line back in buffer */
        if (edrepl()!=OK) {
                return;
        }
        /* insert zero length line at current line */
        if (bufins(editbuf,0)!=OK) {
                return;
        }
```



```
        /* start editing the zero length line */
        edgetln();
        /* update the screen */
        if (edattop()) {
                edsdn(bufln());
                outxy(edxpos(),1);
        }
        else {
                k=outgety();
                bufout(bufln(),k,SCRNL-k);
                outxy(edxpos(),k);
        }
}

/* move cursor right one character.
 * never move the cursor off the current line.
 */

edright()
{
        /* if we are outside the line move right one column */
        if (edxpos() < outgetx()) {
                outxy (min(SCRNW1, outgetx()+1), outgety());
        }
        /* if we are inside a tab move to the end of it */
        else if (edxpos() > outgetx()) {
                outxy (edxpos(), outgety());
        }
        /* move right one character if inside line */
        else if (editp < editpmax) {
                editp++;
                outxy(edxpos(),outgety());
        }
        /* else move past end of line */
        else {
                outxy (min(SCRNW1, outgetx()+1), outgety());
        }
}

/* split the current line into two parts.
 * scroll the first half of the old line up.
 */

edsplit()
{
int p, q;
int k;

        /* indicate that edit buffer has been saved */
        edcflag = NO;
        /* replace current line by the first half of line */
        if (bufatbot()) {
                if (bufins(editbuf, editp) != OK) {
                        return;
                }
        }
        else {
                if (bufrepl(editbuf, editp) != OK) {
                        return;
                }
        }
        /* redraw the first half of the line */
```



```
        p = editpmax;
        q = editp;
        editpmax = editp;
        editp = 0;
        edredraw();
        /* move the second half of the line down */
        editp = 0;
        while (q < p) {
                editbuf [editp++] = editbuf [q++];
        }
        editpmax = editp;
        editp = 0;
        /* insert second half of the line below the first */
        if (bufdn() != OK) {
                return;
        }
        if (bufins(editbuf, editpmax) != OK) {
                return;
        }
        /* scroll the screen up and draw the second half */
        if (edatbot()) {
                edsup(bufln()-SCRNL2);
                outxy(1,SCRNL1);
                edredraw();
        }
```

(Continued on next page)

Screen Oriented Editor

(Listing continued, text begins on page 18)

```
        else {
                k = outgety();
                bufout(bufln(), k+1, SCRNL1-k);
                outxy(1, k+1);
                edredraw();
        }
}

/* move cursor right until end of line or
 * character c found.
 */

edsrch(c) char c;
{
        /* do nothing if at right margin */
        if (editp==editpmax) {
                return;
        }
        /* scan for search character */
        editp++;
        while (editp<editpmax) {
                if (editbuf[editp]==c) {
                        break;
                }
                else {
                        editp++;
                }
        }
        /* reset cursor */
        outxy(edxpos(),outgety());
}

/* move cursor up one line if possible */

edup()
{
int oldx;
```



```
        /* save visual position of cursor */
        oldx=outgetx();
        /* put current line back in buffer */
        if (edrepl()!=OK) {
                return(ERR);
        }
        /* done if at top of buffer */
        if (bufattop()) {
                return(OK);
        }
        /* start editing the previous line */
        if (bufup()!=OK) {
                return(ERR);
        }
        edgetln();
        /* put cursor on this new line as close as
         * possible to where it was on the old line.
         */
        editp=edscan(oldx);
        /* update screen */
        if (edattop()) {
                edsdn(bufln());
                outxy(oldx, 1);
        }
        else {
                outxy(oldx, outgety()-1);
        }
        return(OK);
}

/* delete the current line */

edzap()
{
int k;
        /* delete the line in the buffer */
        if (bufdel()!=OK) {
                return;
        }
        /* move up one line if now at bottom */
        if (bufatbot()) {
                if (bufup()!=OK) {
                        return;
                }
                edgetln();
                /* update screen */
                if (edattop()) {
                        edredraw();
                }
                else {
                        outdelln();
                        outxy(0,outgety()-1);
                }
                return;
        }
        /* start editing new line */
        edgetln();
        /* update screen */
        if (edattop()) {
                edsup(bufln());
                outxy(0,1);
        }
```



```
        else {
                k=outgety();
                bufout(bufln(),k,SCRNL-k);
                outxy(0,k);
        }
}

/* return true if the current edit line is being
 * displayed on the bottom line of the screen.
 */

edatbot()
{
        return(outgety()==SCRNL1);
}

/* return true if the current edit line is being
 * displayed on the bottom line of the screen.
 */

edattop()
{
        return(outgety()==1);
}

/* redraw edit line from index to end of line */
/* reposition cursor */

edredraw()
{
        fmtadj(editbuf,0,editpmax);
        fmtsubs(editbuf,max(0,editp-1),editpmax);
        outxy(edxpos(),outgety());
}

/* return the x position of the cursor on screen */

edxpos()
{
        return(min(SCRNW1,fmtlen(editbuf,editp)));
}


/* fill edit buffer from current main buffer line.
 * the caller must check to make sure the main
 * buffer is available.
 */

edgetln()
{
int k;
        /* put cursor on left margin, reset flag */
        editp=0;
        edcflag=NO;
        /* get edit line from main buffer */
        k=bufgetln(editbuf,MAXLEN);
        if (k>MAXLEN) {
                error("line truncated");
                editpmax=MAXLEN;
        }
        else {
                editpmax=k;
```



```
        }
        fmtadj(editbuf,0,editpmax);
}

/* replace current main buffer line by edit buffer.
 * the edit buffer is NOT changed or cleared.
 * return ERR is something goes wrong.
 */

edrepl()
{
        /* do nothing if nothing has changed */
        if (edcflag==NO) {
                return(OK);
```

```
        }
        /* make sure we don't replace the line twice */
        edcflag=NO;
        /* insert instead of replace if at bottom of file */
        if (bufatbot()) {
                return(bufins(editbuf,editpmax));
        }
        else {
                return(bufrepl(editbuf,editpmax));
        }
}

/* set editp to the largest index such that
 * buf[editp] will be printed <= xpos
 */

edscan(xpos) int xpos;
{
        editp=0;
        while (editp<editpmax) {
                if (fmtlen(editbuf,editp)<xpos) {
                        editp++;
                }
                else {
                        break;
                }
        }
        return(editp);
}

/* scroll the screen up.  topline will be new top line */

edsup(topline) int topline;
{
        if (outhasup()==YES) {
                /* hardware scroll */
                outsup();
                /* redraw bottom line */
                bufout(topline+SCRNL2,SCRNL1,1);
        }
        else {
                /* redraw whole screen */
                bufout(topline,1,SCRNL1);
        }
}

/* scroll screen down.  topline will be new top line */
```



```
edsdn(topline) int topline;
{
        if (outhasdn()==YES) {
                /* hardware scroll */
                outsdn();
                /* redraw top line */
                bufout(topline,1,1);
        }
        else {
                /* redraw whole screen */
                bufout(topline,1,SCRNL1);
        }
}
```

```
/* Screen editor:  output format module
 *
 * Source: ed5.c
 * Version: March 6, 1981.
 */


/* Define variables global to this module */

/* define maximal length of a tab character */

int fmttab;

/* define the current device and device width */

int fmtdev;             /* device -- YES/NO = LIST/CONSOLE */
int fmtwidth;           /* devide width.  LISTW/SCRNW1 */

/* fmtcol[i] is the first column at which
 * buf[i] will be printed.
 * fmtsub() and fmtlen() assume fmtcol[] is valid on entry.
 */

int fmtcol[MAXLEN1];

/* direct output from this module to either the console or
 * the list device.
 */

fmtassn(listflag) int listflag;
{
        if (listflag==YES) {
                fmtdev=YES;
                fmtwidth=LISTW;
        }
        else {
                fmtdev=NO;
                fmtwidth=SCRNW1;
        }
}

/* adjust fmtcol[] to prepare for calls on
 * fmtout() and fmtlen()
 *
 * NOTE:  this routine is needed as an efficiency
 *        measure.  Without fmtadj(), calls on
 *        fmtlen() become too slow.
 */

fmtadj(buf,minind,maxind) char *buf; int minind,maxind;
{
int k;
        /* line always starts at left margin */
        fmtcol[0]=0;
        /* start scanning at minind */
        k=minind;
        while (k<maxind) {
                if (buf[k]==CR) {
                        break;
                }
                fmtcol[k+1]=fmtcol[k]+fmtlench(buf[k],fmtcol[k]);
                k++;
```



```
        }
}

/* return column at which at which buf[i] will be printed */

fmtlen(buf,i) char *buf; int i;
{
        return(fmtcol[i]);
}

/* print buf[i] ... buf[j-1] on current device so long as
 * characters will not be printed in last column.
 */

fmtsubs(buf,i,j) char *buf; int i, j;
{
int k;
        if (fmtcol[i]>=fmtwidth) {
                return;
        }
        outxy(fmtcol[i],outgety());     /* position cursor */
        while (i<j) {
                if (buf[i]==CR) {
                        break;
                }
                if (fmtcol[i+1]>fmtwidth) {
                        break;
                }
                fmtoutch(buf[i],fmtcol[i]);
                i++;
        }
        outdeol();      /* clear rest of line */
}

/* print string which ends with CR or EOS to current device.
 * truncate the string if it is too long.
 */

fmtsout(buf,offset) char *buf; int offset;
{
char c;
int col,k;
        col=0;
        while (c=*buf++) {
                if (c==CR) {
                        break;
                }
                k=fmtlench(c,col);
                if ((col+k+offset)>fmtwidth) {
                        break;
                }
                fmtoutch(c,col);
                col=col+k;
        }
```

(Continued on next page)

Screen Oriented Editor

(Listing continued, text begins on page 18)

```
}

/* return length of char c at column col */

fmtlench(c,col) char c; int col;
{
        if (c==TAB) {
                /* tab every fmttab columns */
```



```
                return(fmttab-(col%fmttab));
        }
        else if (c<32) {
                /* control char */
                return(2);
        }
        else {
                return(1);
        }
}

/* output one character to current device.
 * convert tabs to blanks.
 */

fmtoutch(c,col) char c; int col;
{
int k;
        if (c==TAB) {
                k=fmtlench(TAB,col);
                while ((k--)>0) {
                        fmtievch(' ');
                }
        }
        else if (c<32) {
                fmtdevch('^');
                fmtdevch(c+64);
        }
        else {
                fmtdevch(c);
        }
}

/* output character to current device */

fmtdevch(c) char c;
{
        if (fmtdev==YES) {
                syslout(c);
        }
        else {
                outchar(c);
        }
}

/* output a CR and LF to the current device */

fmtcrlf()
{
        if (fmtdev==YES) {
                syslout(CR);
                syslout(LF);
        }
        else {
                /* kludge: this should be in out module */
                /* make sure out module knows position */
                outxy(0,SCRNL1);
                syscout(CR);
                syscout(LF);
        }
}
```



```
/* set tabs at every n columns */

fmtset(n) int n;
{
        fmttab=max(1,n);
}
```

```
/*
Screen editor:  terminal output module
Source:  ed6.ccc
This file was created by the configuration program:
Version 2:  September 6, 1981.
*/

/*
Define the current coordinates of the cursor.
*/

int outx, outy;

/*
Return the current coordinates of the cursor.
*/

outgetx()
{
        return(outx);
}

outgety()
{
        return(outy);
}

/*
Output one printable character to the screen.
*/

outchar(c) char c;
{
        syscout(c);
        outx++;
        return(c);
}

/*
Position cursor to position x,y on screen.
0,0 is the top left corner.
*/

outxy(x,y) int x,y;
{
        outx=x;
        outy=y;
        syscout(27);
        syscout('=');
        syscout(x+32);
        syscout(y+32);
}

/*
Erase the entire screen.
Make sure the rightmost column is erased.
*/

outclr()
{
int k;
```



```
        k=0;
        while (k<SCRNL) {
                outxy(0,k++);
                outdelln();
        }
        outxy(0,0);
}

/*
Delete the line on which the cursor rests.
Leave the cursor at the left margin.
*/

outdelln()
{
        outxy(0,outy);
        outdeol();
}

/*
Delete to end of line.
Assume the last column is blank.
*/

outdeol()
{
        syscout(27);
        syscout('E');
}

/*
```

```
Return yes if terminal has indicated hardware scroll.
*/

outhasup()
{
        return(YES);
}

outhasdn()
{
        return(YES);
}

/*
Scroll the screen up.
Assume the cursor is on the bottom line.
*/

outsup()
{
        /* auto scroll */
        outxy(0,SCRNL1);
        syscout(10);
}

/*
Scroll screen down.
Assume the cursor is on the top line.
*/

outsdn()
```



```
{
        /* auto scroll */
        outxy(0,0);
        syscout(27);
        syscout('^');
}
```



```
/* Screen editor:  prompt line module
 *
 * Source:  ed7.c
 * Version: March 6, 1981.
 */

/* Define the prompt line data */

char pmtln[MAXLEN];     /* mode */
char pmtfn[SYSFNMAX];   /* file name */


/* put error message on prompt line.
 * wait for response.
 */

pmtmess(s1,s2) char *s1, *s2;
{
int x,y;
        /* save cursor */
        x=outgetx();
        y=outgety();
        outxy(0,0);
        /* make sure line is correct */
        outdelln();
        pmtline1();
        pmtcol1(x);
        /* output error message */
        fmtsout(s1,outgetx());
        fmtsout(s2,outgetx());
        /* wait for input from console */
        syscin();
        /* redraw prompt line */
        pmtline1();
        pmtcol1(x);
        pmtfile1(pmtfn);
        pmtmode1(pmtln);
        /* restore cursor */
        outxy(x,y);
}

/* write new mode message on prompt line */

pmtmode(s) char *s;
{
int x,y;                /* save cursor on entry */
        /* save cursor */
        x=outgetx();
        y=outgety();
        /* redraw whole line */
        outxy(0,0);
        outdelln();
        pmtline1();
        pmtcol1(x);
        pmtfile1(pmtfn);
        pmtmode1(s);
        /* restore cursor */
        outxy(x,y);
}

/* update file name on prompt line */
```



```
pmtfile(s) char *s;
{
int x, y;
        /* save cursor */
        x=outgetx();
        y=outgety();
        /* update whole line */
        outxy(0,0);
        outdelln();
        pmtline1();
        pmtcol1();
        pmtfile1(s);
        pmtmode1(pmtln);
        /* restore cursor */
        outxy(x,y);
}

/* change mode on prompt line to edit: */

pmtedit()
{
        pmtmode("edit:");
}

/* update line and column numbers on prompt line */

pmtline()
{
int x,y;
        /* save cursor */
        x=outgetx();
        y=outgety();
        /* redraw whole line */
        outxy(0,0);
        outdelln();
        pmtline1();
        pmtcol1(x);
        pmtfile1(pmtfn);
        pmtmode1(pmtln);
        /* restore cursor */
        outxy(x,y);
}

/* update just the column number on prompt line */

pmtcol()
{
int x,y;
        /* save cursor */
        x=outgetx();
        y=outgety();
        /* update column number */
        pmtcol1(x);
        /* update cursor */
        outxy(x,y);
}

/* update mode.  call getcmnd() to write on prompt line */

pmtcmnd(mode,buffer) char *mode, *buffer;
{
int x,y;
```



```
        /* save cursor */
        x=outgetx();
        y=outgety();
        pmtmode1(mode);
        /* user types command on prompt line */
        getcmnd(buffer,outgetx());
        /* restore cursor */
}

/* update and print mode */

pmtmode1(s) char *s;
{
int i;
```

(Continued on next page)

Screen Oriented Editor

(Listing continued, text begins on page 18)

```
        outxy(40,0);
        fmtsout(s,40);
        i=0;
        while (pmtln[i++]=*s++) {
                ;
        }
}

/* print the file name on the prompt line */

pmtfile1(s) char *s;
{
int i;
        outxy(25,0);
        if (*s==EOS) {
                fmtsout("no file",25);
        }
        else {
                fmtsout(s,25);
        }
        i=0;
        while (pmtfn[i++]=*s++) {
                ;
        }
}

/* print the line number on the prompt line */

pmtline1()
{
        outxy(0,0);
        fmtsout("line: ",0);
        putdec(bufln(),5);
}


/* print column number of the cursor */

pmtcol1(x) int x;
{
        outxy(12,0);
        fmtsout("column: ",12);
        putdec(x,3);
}
```

```
ed8.c                September 8, 1981                page 1

/* Screen editor:  operating system module
 *
 * Source:  ed8.c
 * Version: June 19, 1981.
 */

/* all calls to the operating system are made here.
 * only this module and the assembler libraries will have to
 * rewritten for a new operating system.
 */

/* the routines syscstat(), syscin() and syscout() come in
 * two flavors:  CP/M version 2.2 and CP/M version 1.4.
 * Comment out which ever you don't use.
 */

/* CP/M 2.2 versions of syscstat(), syscin(), syscout() */

/* return -1 if no character is ready from the console.
 * otherwise, return the character.
 */

syscstat()
{
        return(cpm(6,-1));
}

/* wait for next character from the console.
 * do not echo it.
 */

syscin()
{
int c;
        while ((c=cpm(6,-1))==0) {
                ;
        }
        return(c);
}

/* print character on the console */

syscout(c) char c;
{
        cpm(6,c);
        return(c);
}

/* CP/M 1.4 versions of syscstat(), syscin(), syscout() */

/* start comment out ----------
#asm
;
;       bios(n,bc)
;
;       call the n'th bios routine.
;       bc is put into the bc-register as a parameter.
;       returns whatever bios puts into the a-reg.
;
QZBIOS:
        POP     H       ;get return address
        POP     B       ;    bc
ed8.c                September 8, 1981                page 2

        POP     D       ;    n
        PUSH    D       ;restore stack the way it was
        PUSH    B
        PUSH    H
        LHLD    1       ;point hl at start of BIOS jump table
        DCX     H
        DCX     H
        DCX     H
        MOV     A,E     ;put index into jump table into de
        ADD     A
        ADD     E
        MVI     D,0
        MOV     E,A
        DAD     D       ;point hl at proper entry in jump table
        PUSH    H       ;push call address
        LXI     H,QZBIOS1 ;point hl at return address
        XTHL            ;push return address, get call address
        PCHL            ;call proper BIOS routine
QZBIOS1:
        MOV     L,A     ;put return code from BIOS into hl
        MVI     H,0
        RET             ;go back to small-c
#endasm

syscstat()
{
        if (bios(2,0)==255) {
                return(-1);
        }
        else {
                return(0);
        }
}

syscin()
{
        return(bios(3,0));
}

syscout(c) char c;
{
        bios(4,c);
        return(c);
}
---------- end comment out */


/* print character on the printer */

syslout(c) char c;
{
        cpm(5,c);
        return(c);
}

#asm
;
; sysend()
;
; return address of last usable memory location.
;
QZSYSEND:
ed8.c                September 8, 1981                page 3

        LHLD    6       ;get address of BDOS-1
        DCX     H       ;
        RET             ;return
#endasm

/* open a file */

sysopen(name,mode) char *name, *mode;
```

```
{
int file;
        if ((file=fopen(name,mode))==0) {
                return(ERR);
        }
        else {
                return(file);
        }
}

/* close a file */

sysclose(file) int file;
{
        /* fclose doesn't reliably return OK */
        fclose(file);
        return(OK);
}

/* read next char from file */

sysrdch(file) int file;
{
int c;
        if ((c=getc(file))==-1) {
                return(EOF);
        }
        else {
                return(c);
        }
}

/* write next char to file */

syspshch(c,file) char c; int file;
{
        if (putc(c,file)==-1) {
                error("disk write failed");
                return(ERR);
        }
        else {
                return(c);
        }
}

/* read one char from END of file */

syspopch(file) int file;
{
        error("syspopch() not implemented");
        return(ERR);
}

/* check file name for syntax */
```



```
syschkfn(args) char *args;
{
        return(OK);
}

/* copy file name from args to buffer */

syscopfn(args,buffer) char *args, *buffer;
{
int n;
        n=0;
        while (n< SYSFNMAX-1) {
                if (args[n]==EOS) {
                        break;
                }
                else {
                        buffer[n]=args[n];
                        n++;
                }
        }
        buffer[n]=EOS;
}

/* move a block of n bytes down (towards HIGH addresses).
 * block starts a source and the first byte goes to dest.
 * this routine is only called from bufmovdn() as follows
 *      sysmovdn( n=to-from+1, dest=to+length, source=to);
 */

#asm
QZSYSMOVDN:
        POP     H       ;get return address
        POP     B       ;BC = source
        POP     D       ;DE = dest
        XTHL            ;HL = n, restore return address
        JMP     SYSMOVDN2       ;go enter loop
;
; this code is 15 times faster than the c-code it replaces.
SYSMOVDN1:
        LDAX    B       ;*dest-- = *source--
        STAX    D
        DCX     B
        DCX     D
        DCX     H       ;while ((n--)>0)
SYSMOVDN2:
        MOV     A,L
        ORA     H
        JNZ     SYSMOVDN1
;
        XTHL            ;HL = return address
        PUSH    H       ;restore stack
        PUSH    H
        PUSH    H
        RET             ;return
#endasm

/* move a block of n bytes up (towards LOW addresses).
 * the block starts at source and the first byte goes to dest.
 * this routine is called only from bufmovup() as follows:
 *      sysmovup( n=to-from+1, dest=from-length, source=from);
 */
```



```
#asm
QZSYSMOVUP:
        POP     H       ;get return address
        POP     B       ;BC = source
        POP     D       ;DE = dest
        XTHL            ;HL = n, save return address
        JMP     SYSMOVUP2       ;go enter loop
;
; this code is 15 times faster than the c-code it replaces.
SYSMOVUP1:
        LDAX    B       ;*dest++ = *source++
        STAX    D
        INX     B
        INX     D
        DCX     H       ;while ((n--)>0)
SYSMOVUP2:
        MOV     A,L
        ORA     H
        JNZ     SYSMOVUP1
;
        XTHL            ;HL = return address
        PUSH    H       ;restore stack
        PUSH    H
        PUSH    H
        RET             ;return
#endasm
```



```
/* Screen editor:  general utilities
 *
 * Source:  ed9.c
 * Version: May 3, 1981.
 */

/* convert lower case to upper case */

toupper(c) int c;
{
        if ((c<'a')||(c>'z')) {
                return(c);
        }
        else {
                return(c-32);
        }
}

/* convert upper case to lower case */

tolower(c) int c;
{
        if ((c<'A')||(c>'Z')) {
                return(c);
        }
        else {
                return(c+32);
        }
}

/* return: is first token in args a number ? */
/* return value of number in *val            */

number(args,val) char *args; int *val;
{
char c;
        c=*args++;
```

(Continued on next page)

Screen Oriented Editor

(Listing continued, text begins on page 18)

```
        if ((c<'0')|(c>'9')) {
                return(NO);
        }
        *val=c-'0';
        while (c=*args++) {
                if ((c<'0')|(c>'9')) {
                        break;
                }
                *val=(*val*10)+c-'0';
        }
        return(YES);
}

/* convert character buffer to numeric */

ctoi(buf,index) char *buf; int index;
{
int k;
        while ( (buf[index]==' ') |
                (buf[index]==TAB) ) {
                index++;
        }
        k=0;
        while ((buf[index]>='0')&(buf[index]<='9')) {
                k=(k*10)+buf[index]-'0';
ed9.c                   May 11, 1981                    page 2

                index++;
        }
        return(k);
}


/* return maximum of m,n */

max(m,n) int m,n;
{
        if (m>=n) {
                return(m);
        }
        else {
                return(n);
        }
}

/* return minimum of m,n */

min(m,n) int m,n;
{
        if (m<=n) {
                return(m);
        }
        else {
                return(n);
        }
}

/* put decimal integer n in field width >= w.
 * left justify the number in the field.
 */

putdec(n,w) int n,w;
{
char chars[10];
int i,nd;
        nd=itoc(n,chars,10);
        i=0;
        while (i<nd) {
                syscout(chars[i++]);
        }
        i=nd;
        while (i++<w) {
                syscout(' ');
        }
}

/* convert integer n to character string in str */

itoc(n,str,size) int n; char *str; int size;
{
int absval;
int len;
int i,j,k;
        absval=abs(n);
        /* generate digits */
        str[0]=0;
        i=1;
        while (i<size) {
                str[i++]=(absval%10)+'0';
ed9.c                   May 11, 1981                    page 3

                absval=absval/10;
                if (absval==0) {
                        break;
                }
        }
        /* generate sign */
        if ((i<size)&(n<0)) {
                str[i++]='-';
        }
        len=i-1;
        /* reverse sign, digits */
        i--;
        j=0;
        while (j<i) {
                k=str[i];
                str[i]=str[j];
                str[j]=k;
                i--;
                j++;
        }
        return(len);
}

/* return absolute value of n */

abs(n) int n;
{
        if (n<0) {
                return(-n);
        }
        else {
                return(n);
        }
}

/* system error routine */

syserr(s) char *s;
{
        pmtmess("system error: ",s);
}

/* user error routine */

error(s) char *s;
{
        pmtmess("error: ",s);
}

/* disk error routine */

diskerr(s) char *s;
{
        pmtmess("disk error: ",s);
}

/* read the next line of the file into
 * the buffer of size n that p points to.
 * Successive calls to readline() read the file
 * from front to back.
 */
ed9.c                   May 11, 1981                    page 4

readline(file,p,n) int file; char *p; int n;
{
int c;
int k;
        k=0;
        while (1) {
                c=sysrdch(file);
                if (c==ERR) {
                        return(ERR);
                }
                if (c==EOF) {
                        /* ignore line without CR */
                        return (EOF);
                }
                if (c==CR) {
                        return(k);
                }
                if (k<n) {
                        /* move char to buffer */
                        *p++=c;
                }
                /* always bump count */
                k++;
        }
}

/* push (same as write) line to end of file.
```

```
 * line is in the buffer of size n that p points to.
 * lines written by this routine may be read by
 * either readline() or popline().
 */

pushline(file,p,n) int file; char *p; int n;
{
        /* write all but trailing CR */
        while ((n--)>0) {
                if (syspshch(*p++,file)==ERR) {
                        return(ERR);
                }
        }
        /* write trailing CR */
        return(syspshch(CR,file));
}

/* pop a line from the back of the file.
 * the line should have been pushed using pushline().
 */

popline(file,p,n) int file; char *p; int n;
{
int c;
int k, kmax, t;
        /* first char must be CR */
        c=syspopch(file);
        if (c==EOF) {
                /* at START of file */
                return(EOF);
        }
        if (c==CR) {
                /* put into buffer */
                *p++=CR;
                k=1;
ed9.c                   May 11, 1981                    page 5

        }
        else {
                syserr("popline: missing CR");
                return(ERR);
        }
        /* pop line into buffer in reverse order */
        while (1) {
                c=syspopch(file);
                if (c==ERR) {
                        return(ERR);
                }
                if (c==EOF) {
                        break;
                }
                if (c==CR) {
                        /* this ends ANOTHER line */
                        /* push it back                */
                        if (syspshch(CR,file)==ERR) {
                                return(ERR);
                        }
                        break;
                }
                /* non-special case */
                if (k<n) {
                        /* put into buffer */
                        *p++=c;
                }
                /* always bump count */
                k++;
        }
        /* remember if we truncated the line */
        kmax=k;
        /* reverse the buffer */
        k=min(k,n-1);
        t=0;
        while (k>t) {
                /* swap p[t], p[k] */
                c=p[k];
                p[k]=p[t];
                p[t]=c;
                k--;
                t++;
        }
        return(kmax);
}
```

```
ed10.c                  May 11, 1981                    page 1

/* Screen editor:  buffer module
 *
 * Source:  ed10.c
 * Version: April 7, 1981.
 */

/* Define the variables global to this module.
 * Buffer must be declared after all other variables
 * of the entire program.
 * Note: buffer must have nonzero dimension.
 */

int bufcflag;           /* main buffer changed flag */
char *bufp;             /* start of current line */
char *bufpmax;          /* end of last line */
char *bufend;           /* last byte of buffer */
int bufline;            /* current line number */
int bufmaxln;           /* number of lines in buffer */
char buffer[1];         /* start of buffer */

/* This code is built around several invariant
 * assumptions:
 * First, the last line is always completely empty.
 * When bufp points to the last line there is NO
 * CR following it.
 * Second, bufp points to the last line if and only if
 * bufline==bufmaxln+1.
 * Third, bufline is always greater than zero.
 * Line zero exists only to make scanning for the
 * start of line one easier.
 */


/* Clear the main buffer */

bufnew()
{
        /* point past line 0 */
        bufp=bufpmax=buffer+1;
        /* point at last byte of buffer */
        /* allow space for stack */
        bufend=sysend()-1000;
        /* at line one. no lines in buffer */
        bufline=1;
        bufmaxln=0;
        /* line zero is always a null line */
        buffer[0]=CR;
        /* indicate no need to save file yet */
        bufcflag=NO;
}

/* return current line number */

bufln()
{
        return(bufline);
}

/* return YES if the buffer (i.e., file) has been
 * changed since the last time the file was saved.
 */
ed10.c                  May 11, 1981                    page 2

bufchng()
{
        return(bufcflag);
}

/* the file has been saved.  clear bufcflag. */

bufsaved()
{
        bufcflag=NO;
}

/* return number of bytes left in the buffer */

buffree()
{
        return(bufend-bufp);
}

/* Position buffer pointers to start of indicated line */

bufgo(line) int line;
{
        /* put request into range. prevent extension */
        line=min(bufmaxln+1,line);
        line=max(1,line);
        /* already at proper line? return. */
        if (line==bufline) {
                return(OK);
        }
        /* move through buffer one line at a time */
        while (line<bufline) {
                if (bufup()==ERR) {
                        return(ERR);
                }
        }
        while (line>bufline) {
                if (bufdn()==ERR) {
                        return(ERR);
```

(Continued on next page)

Screen Oriented Editor

(Listing continued, text begins on page 18)

```
                }
        }
        /* we have reached the line we wanted */
        return(OK);
}

/* move one line closer to front of buffer, i.e.,
 * set buffer pointers to start of previous line.
 */

bufup()
{
char *oldbufp;
        oldbufp=bufp;
        /* can't move past line 1 */
        if (bufattop()) {
                return(OK);
        }
        /* move past CR of previous line */
        if (*--bufp!=CR) {
                syserr("bufup: missing CR");
                bufp=oldbufp;
                return(ERR);
```



```
        }
        /* move to start of previous line */
        while (*--bufp!=CR) {
                ;
        }
        bufp++;
        /* make sure we haven't gone too far */
        if (bufp<(buffer+1)) {
                syserr("bufup: bufp underflow");
                bufp=oldbufp;
                return(ERR);
        }
        /* success!  we ARE at previous line */
        bufline--;
        return(OK);
}

/* Move one line closer to end of buffer, i.e.,
 * set buffer pointers to start of next line.
 */

bufdn()
{
char *oldbufp;
        oldbufp=bufp;
        /* do nothing silly if at end of buffer */
        if (bufatbot()) {
                return(OK);
        }
        /* scan past current line and CR */
        while (*bufp++!=CR) {
                ;
        }
        /* make sure we haven't gone too far */
        if (bufp>bufpmax) {
                syserr("bufdn: bufp overflow");
                bufp=oldbufp;
                return(ERR);
        }
        /* success! we are at next line */
        bufline++;
        return(OK);
}

/* Insert a line before the current line.
 * p points to a line of length n to be inserted.
 * Note: n does not include trailing CR.
 */

bufins(p,n) char *p; int n;
{
int k;
        /* make room in the buffer for the line */
        if (bufext(n+1)==ERR) {
                return(ERR);
        }
        /* put the line and CR into the buffer */
        k=0;
        while (k<n) {
                *(bufp+k)=*(p+k);
                k++;
        }
```



```
        *(bufp+k)=CR;
        /* increase number of lines in buffer */
        bufmaxln++;
        /* special case: inserting a null line at
         * end of file is not a significant change.
         */
        if ((n==0)&(bufnrbot())) {
                ;
        }
        else {
                bufcflag=YES;
        }
        return(OK);
}

/* delete the current line */

bufdel()
{
        return(bufdeln(1));
}

/* delete n lines, starting with the current line */

bufdeln(n) int n;
{
int oldline;
int k;
char *oldbufp;
        /* remember current buffer parameters */
        oldline=bufline;
        oldbufp=bufp;
        /* scan for first line after deleted lines */
        k=0;
        while ((n--)>0) {
                if (bufatbot()) {
                        break;
                }
                if (bufdn()==ERR) {
                        bufline=oldline;
                        oldbufp=bufp;
                        return(ERR);
                }
                k++;
        }
        /* compress buffer.  update pointers */
        bufmovup(bufp,bufpmax-1,bufp-oldbufp);
        bufpmax=bufpmax-(bufp-oldbufp);
        bufp=oldbufp;
        bufline=oldline;
        bufmaxln=bufmaxln-k;
        bufcflag=YES;
        return(OK);
}

/* replace current line with the line that
 * p points to.  The new line is of length n.
 */

bufrepl(p,n) char *p; int n;
{
int oldlen, k;
```



```
char *nextp;
        /* do not replace null line.  just insert */
        if (bufatbot()) {
                return(bufins(p,n));
        }
        /* point nextp at start of next line */
        if (bufdn()==ERR) {
                return(ERR);
        }
        nextp=bufp;
        if (bufup()==ERR) {
                return(ERR);
        }
        /* allow for CR at end */
        n=n+1;
        /* see how to move buffer below us;
         * up, down, or not at all.
         */
        oldlen=nextp-bufp;
        if (oldlen<n) {
                /* move buffer down */
                if (bufext(n-oldlen)==ERR) {
                        return(ERR);
                }
                bufpmax=bufpmax+n-oldlen;
        }
        else if (oldlen>n) {
```

```
                /* move buffer up */
                bufmovup(nextp,bufpmax-1,oldlen-n);
                bufpmax=bufpmax-(oldlen-n);
        }
        /* put new line in the hole we just made */
        k=0;
        while (k<(n-1)) {
                bufp[k]=p[k];
                k++;
        }
        bufp[k]=CR;
        bufcflag=YES;
        return(OK);
}

/* copy current line into buffer that p points to.
 * the maximum size of that buffer is n.
 * return k=length of line in the main buffer.
 * if k>n then truncate n-k characters and only
 * return n characters in the caller's buffer.
 */

bufgetln(p,n) char *p; int n;
{
int k;
        /* last line is always null */
        if (bufatbot()) {
                return(0);
        }
        /* copy line as long as it not too long */
        k=0;
        while (k<n) {
                if (*(bufp+k)==CR) {
                        return(k);
                }
```



```
                *(p+k)=*(bufp+k);
                k++;
        }
        /* count length but move no more chars */
        while (*(bufp+k)!=CR) {
                k++;
        }
        return(k);
}

/* move buffer down (towards HIGH addresses) */

bufmovdn(from,to,length) char *from, *to; int length;
{
        /* this code needs to be very fast.
         * use an assembly language routine.
         */
        sysmovdn(to-from+1,to+length,to);
}

/* the call to sysmovdn() is equivalent to the following code:

int k;
        k=to-from+1;
        while ((k--)>0) {
                *(to+length)=*to;
                to--;
        }

 */

/* move buffer up (towards LOW addresses) */

bufmovup(from,to,length) char *from, *to; int length;
{
        /* this code must be very fast.
         * use an assembly language routine.
         */
        sysmovup(to-from+1,from-length,from);
}

/* the call to sysmovup() is equivalent to the following code:

int k;
        k=to-from+1;
        while ((k--)>0) {
                *(from-length)=*from;
                from++;
        }

 */

/* return true if at bottom of buffer.
 * NOTE 1: the last line of the buffer is always null.
 * NOTE 2: the last line number is always bufmaxln+1.
 */

bufatbot()
{
        return(bufline>bufmaxln);
}
```



```
/* return true if at bottom or at the last
 * real line before the bottom.
 */

bufnrbot()
{
        return(bufline>=bufmaxln);
}

/* return true if at top of buffer */

bufattop()
{
        return(bufline==1);
}

/* put nlines lines from buffer starting with
 * line topline at position topy of the screen.
 */

bufout(topline,topy,nlines) int topline, topy, nlines;
{
int l,p;
        /* remember buffer's state */
        l=bufline;
        p=bufp;
        /* write out one line at a time */
        while ((nlines--)>0) {
                outxy(0,topy++);
                bufoutln(topline++);
        }
        /* restore buffer's state */
        bufline=l;
        bufp=p;
}

/* print line of main buffer on screen */

bufoutln(line) int line;
{
        /* error message does NOT go on prompt line */
        if (bufgo(line)==ERR) {
                fmtsout("disk error: line deleted",0);
                outdeol();
                return;
        }
        /* blank out lines below last line of buffer */
        if (bufatbot()) {
                outdeol();
        }
        /* write one formatted line out */
        else {
                fmtsout(bufp,0);
                outdeol();
        }
}

/* simple memory version of bufext.
 * create a hole in buffer at current line.
 * length is the size of the hole.
 */
```



```
bufext(length) int length;
{
        /* make sure there is room for more */
        if ((bufpmax+length)>=bufend) {
                error("main buffer is full");
                return(ERR);
        }
        /* move lines below current line down */
        bufmovdn(bufp,bufpmax-1,length);
        bufpmax=bufpmax+length;
        return(OK);
}
```

End Listing