Text Editor Prevents "Starvation" and Programming Hassles

By Donald Fitchhorn

MITS

Computer Notes, October 1977.

Why I wrote and editor

My main objective in writing an editor was to complement DISK EXTENDED BASIC's built-in EDIT feature. This feature is invaluable if the location of a needed change is already known. But what about our friend Bob's problem? He knew WHAT the problem was but not WHERE to find it. What he needs is a program that will search through the entire file until it finds what he wants. Since there isn't such an editor in Disk Extended BASIC, I wrote my own in BASIC so that it can be easily changed.

The search command and others like it where the beginnings of my editor, which now has 14 commands. But before we get into an explanation of this editor, let's review the definition of an editor.

What is an editor?

An editor is a program which permits the addition and deletion of lines and chracters in one file to make another file. The second file is the new up-to-date file, and the first file is retained as the backup file. Some editors permit the creation of new files and/or use multiple input files. An editor can be as extensive or as minimal as is necessary for an application.

PROGRAM FILES

The EDIT program works on ASCII program files. A program file is any file that looks like a program to BASIC. (see EXAMPLE 1.) BASIC doesn't care if the whole file is REMark statements; it's only concerned with whether or not there is a line number at the beginning of each line. (See EXAMPLE 2.)

(NOTE: The EDIT program cannot handle files saved in binary. Save all files in ASCII, i.e. `SAVE "FILENAME", 0, A`)

`EXAMPLE #1`

```
10 'THIS IS A PROGRAM FILE
20 FOR I = 1 TO 100000
30 PRINT I;RND(I)*1000;I*RND(I)
40 ' THIS PRINTS SOME NUMBERS
```

```
50 NEXT
60 END
```

EXAMPLE #2

```
10 'This is a document program file
20 'here is the
30 'text of the
40 'document !
50 'this is the end.
```

EXAMPLE #3
LOWER CASE IS USER TYPED

```
run"edit
EDIT -- VERSION 1.0
INPUT FILE NAME?time
>r
EOF1
CLEARING..........
>/1
5 LPRINT"MINUTES","HUNDREDTHS",,"MINUTES","HUNDREDTHS"
10 FORI=6TO30STEP-1
20 J=INT(I/60*100)
21 K=INT((I/30)/60*100)
30 LPRINTI,J,,I-30,K
40 NEXT
>gJ
20 J
>cL2
30 L2=INT(I/60*100)
>gJ
30 LPRINTI,J
>cL2
30 LPRINTI,L2,,I-30,K
>gJ
EOB
>x
BACKUP FILE NAME?time.bak
OK
load"time
OK
list
5 LPRINT"MINUTES","HUNDREDTHS",,"MINUTES","HUNDREDTHS"
10 FOR I=60TO30STEP-1
20 J=INT(I/60*100)
21 K=INT((I-30)/60*100)
30 LPRINTI,J,,I-30,K
```

```
40 NEXT
OK
```

Saving documents as programs (EXAMPLE 2 format) allows them to be loaded with BASIC. This allows BASIC to be used to alter, delete, and add lines. Of course, line numbers on the finished document may not be wanted, so a short program that reads the file and PRINTs `MID$(LINE$, INSTR(LINE$,"'")+1` will print everything to the right of the (‘). Be sure to use `LINE INPUT` instead of `INPUT` when reading up `LINE$` to prevent truncation because of commas in the text. The (’)‘s in EXAMPLE 2 are necessary if the program file is to be loaded and saved by BASIC. Without the (’), BASIC will modify the text line.

BASIC won’t allow lines to be moved around within the file without retyping each line. But with the EDIT program, line numbers can be changed to whatever is wanted. When the edited file is loaded into BASIC, BASIC will put the lines in numerical order.

Internal structure

The EDIT program maintains a double-linked list in memory. Each line (`L1$(X)`) has a pointer to the previous line [`M1(X,0)`] and to the next line [`M1(X,1)`] added to it when it is read in. The array (L1$) that the lines are kept in is divided into two parts – ACTIVE CELLS, which have data in them, and INACTIVE CELLS, available for use as data lines. Deleted lines are linked into the INACTIVE CELLS from the ACTIVE CELLS. Inserted lines are written into the first available INACTIVE CELL and then linked into the ACTIVE CELLS. Pointers are maintained for FIRST ACTIVE CELL (LN), FIRST INACTIVE CELL (IN), DOT or position within cell (H) and CELL that DOT is in (J).

Commands

My editor, like many others, uses a single letter to select a command. For example, `A` will advance DOT one line. Most commands may be preceded by a number or a slash (/) to indicate that hey should be executed more than once. 13A will advance DOT 13 lines. OA will position DOT at the beginning of the current line. /A will advance DOT to the end of the page. All commands that allow a prefix will default to one if none is specified. The following is an explanation of the commands. (See TABLE 1 for a list of commands in alphabetical order. See TABLE 2 for a list divided into four main groups).

TABLE #1

COMMAND	DESCRIPTION	OPERATES ON: LINE	CHAR	DOT	FILE	ALLOWED: 0	#	-#	/
A	ADVANCE			*		*	*	*	*
B	BEGINNING			*					
C	CHANGE		*						
D	DELETE		*				*		
E	END			*					
G	GET		*				*		
I	INSERT	*	*						
J	JUMP			*		*	*	*	
K	KILL	*					*		*
L	LIST	*					*		*
N	NEXT				*				
R	READ				*				
V	VERIFY	*							
X	EXIT				*				

```
TABLE #2
COMMANDS THAT AFFECT LINES
I - Inserts lines until, backslash (\) is entered.
K - Kills entire line. #K & /K are legal.
L - Lists entire line. #L & /L are legal.
V - Prints the current line up to DOT.
    (Verify's the position of DOT).

COMMANDS THAT AFFECT CHARACTERS
C - Changes last character string gotten with G command to
    something else. Use: Ghere
                         Cthere
    will change here to there.
D - Delete a character. #D legal.
G - Get string. Searches for occurrence of string.
    Use: 3Gstuff
    Finds third occurrrence of stuff. #G legal.
I - Insert characters at DOT. Use: Iabcde
    Will insert abcde in current line at DOT.

COMMANDS THAT MOVE DOT
A - Moves DOT forward or back the number of lines
    specified. 0A, #A, -#A, /A(same as E) legal.
B - Moves DOT to beginning of page .
E - Moves DOT to end of page. (same as /A)
J - Moves DOT forward or back the number of characters
    specified. 0J, #J, -#J legal.

COMMANDS THAT READ AND WRITE THE FILE
N - Writes out current page and reads in next page.
R - Read in a new page.
X - Writes out current page then reads and writes until
    end of file. Closes and renames files.
```

How to use edit's commands

The commands in EDIT are broken up into four groups, as shown in TABLE 2. The use of these commands will be explained in that order.

The commands that affect lines will work on one line at a time. Or, in the case of K & L, they may be preceded by a number or slash (/) to indicate that they are to be performed on several succeeding lines. TO begin insertion of lines, type I `<return>` and then the lines to insert. To tell EDIT that the last line to be inserted has been entered, type backslash ().

Of the commands that affect characters I & C cannot have a # prefix. #D

deletes # characters to the right of DOT. G & C work together to allow gettings a string and then changing it to something else. G moved DOT ahead of the Nth occurrence of the string. Then C can be used to change th nth occurrence to a new string. It works like C, except instead of changing one string for another, it inserts a new string ahead of DOT.

The commands that move DOT are A, B, E, and J. B & E require no other specifiers. They simply move DOT to the beginning or end of the current page. A & J move DOT forward or back a specified number of lines or characters.

The ommands N, R, and X read and write the files. R reads the input file until EOF, until it has read 50 lines, or 2000 characters. It then clears and resets the INACTIVE CELLS. N writes out the current page and then executes and R (reads in the next page). X does a series of N's until the input file is EOF. Then it closes the files and renames them by giving the input file a backup name and the output file the input file's old name.

The best way to learn to use EDIT is to load it in and try a few commands. (See EXAMPLE 3). Once you get the hang of it, the power and versatility will be well worth the time it took t type it in.

The other side of the coin

All of the above is really wonderful isn't it? But this program is not without its limitations.

1. The commands may or may not work the way the user expects them to. This is a typical problem in any new program because the commands take some getting used to. If, after trying it for a while, the user wants a command to work differently or wants to use another command, just remember that the program is written in BASIC, so modifications are easy.

2. The program allows workin on large files by breaking the file into pages. This works out well except for one thing. No matter what the user does, string space is used. Eventually, all available space will be used.

   At that time, BASIC has to look through all of the string space, shuffling things around and freeing up no-longer-used bytes so that the program will have some more space. This is commonly referred to as GARBAGE COLLECTING. IT can happen at the most unlikely of times and can take as long as five minutes. Unfortunately to the unsuspecting user, it looks as though the program has bombed BASIC out because CTRL-C and RESET don't help solve the problem. But patience is rewarded and the program does come back to life.

Modifications and improvements

I will leave these up to the reader because they are easy to make. For example, suppose a command is needed to exit the program gracefully without making any changes. Follow these steps:

1. Pick a command character. How about Q for quit?
2. Alter line 250 to reflect where the program should go if it sees a Q. Let's make this 1000.
3. Put in the necessary code to perform the command. `basic 1000 CLOSE:CLEAR 200:END`
4. Save the new program.