CONGRATULATIONS ON YOUR PURCHASE OF THE PLATINUM SOFTWARE WORKSAVER

40 " "
for NON ECB after CL

You have just made a major upgrade of your COLOR COMPUTER SYSTEM. The WORKSAVER is more than just a software program that runs ON the COLOR COMPUTER, it's a keyboard system that runs WITH the COLOR COMPUTER. The COLOR COMPUTER is a powerful small home computer, but if you've tried writing programs, you've discovered that the home computer isn't as useful as the literature would like you to believe. Editing programs is tedious, entering data requires perfection or a lot of retyping, and all in all, whatever it was you set out to make easier by using your computer, probably turned out having its own set of hassles. The most typical of these relate to input data updates, corrections and redundant menu answering.

THE WORKSAVER ADDS THE SCRATCH PAD CONVENIENCE THAT IS MISSING, NOT ONLY FROM THE COLOR COMPUTER, BUT FROM MOST HOME COMPUTERS ON THE MARKET TODAY.

GETTING STARTED

We have concentrated on applying human factors engineering to the design of our overlay so that the WORKSAVER functions would be obvious from the start. But we reached a point where we gave up and wrote this manual.

Learning by doing is the best teacher; therefore, we have included examples with every description.

The first thing you need to know is that the WORKSAVER is an extension to the BASIC ROM(s). Once it is loaded, it becomes part of your computer's operating system, and you never have to turn it off. Any program written in BASIC for the COLOR COMPUTER can be edited and run with the WORKSAVER. The only difference you should detect is a different cursor and a whole lot more convenience.

To get started you'll need to load the WORKSAVER by typing:

CLOADM (and pressing the enter key).

While the program is loading, you will be greeted with a cover screen. After the program is loaded enter

EXEC (and press the enter key).

If you have EXTENDED BASIC, you will be given the option to relocate the WORKSAVER by responding to the prompt,

ENTER RELOCATION ADDRESS .....

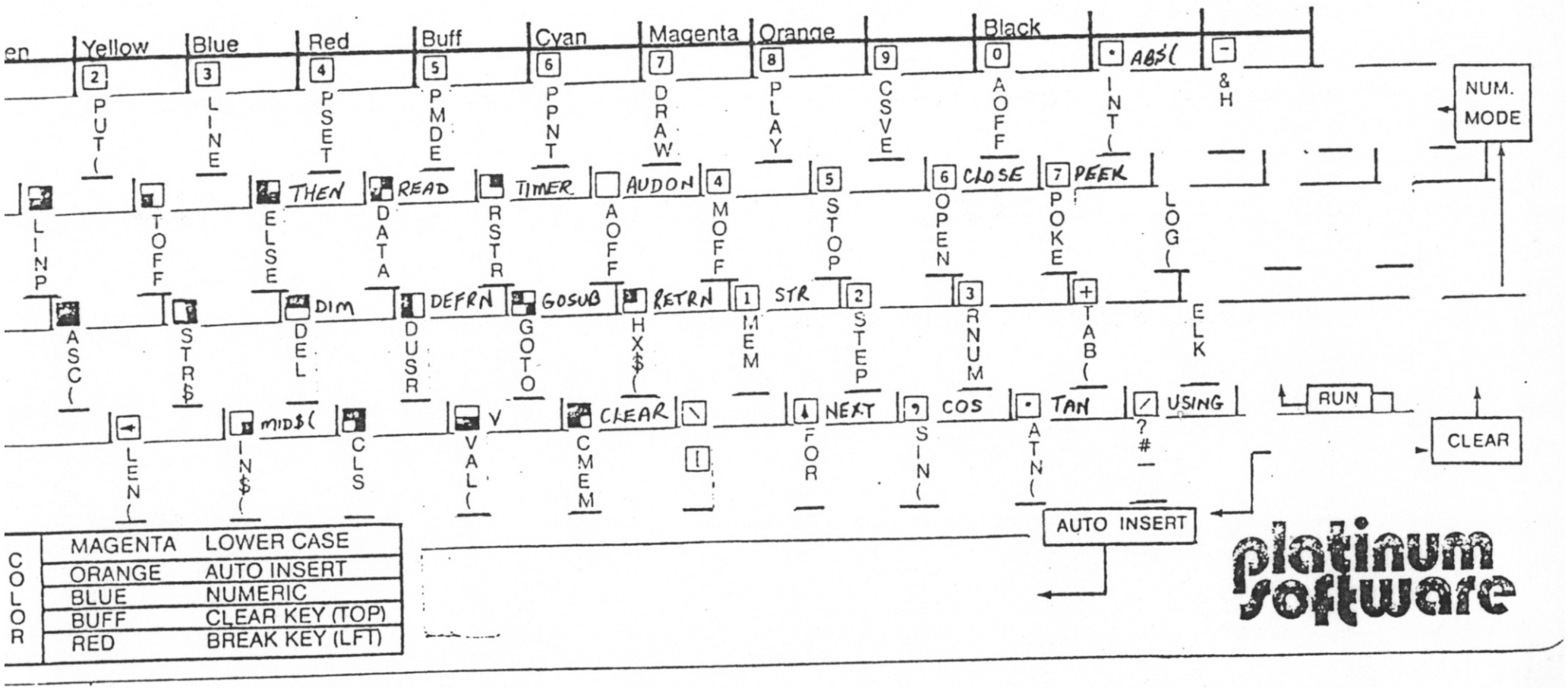
PRESS (ENTER) FOR DEFAULT

The WORKSAVER assumes you wish for it to run at the highest possible memory address below HEX 8000 (for COLOR BASIC the WORKSAVER is automatically moved to those locations). If you are running a 16K or 32K computer, then this is the safest place for the WORKSAVER, and you should ignore the RELOCATE prompt BY PRESSING THE ENTER KEY. On the other hand, if you have a RAM extension card, such as the EXATRON 32K 'THING', then you'll want the WORKSAVER relocated above address HEX C000. To do this, type in the DECIMAL (we know we've been talking HEX but enter DECIMAL address where you want the WORKSAVER to start (the WORKSAVER version A1 requires 1992 bytes less than 2K).

After you press the 'ENTER' key, you will be greeted with the familiar 'OK' statement.

INITIALIZATION EXAMPLE:

1. IF YOU HAVE COLOR BASIC, THE WORKSAVER WILL AUTOMATICALLY RELOCATE AFTER YOU ENTER 'EXEC' AND PRESS THE ENTER KEY. THE SCREEN WILL CLEAR AND YOU CAN START USING THE WORKSAVER.
2. IF YOU HAVE AN EXTENDED BASIC COMPUTER WITH 16K, THEN AFTER TYPING 'EXEC' AND PRESSING THE ENTER KEY ONCE, ALL YOU DO IS PRESS THE ENTER KEY AGAIN


Yellow
Blue
Red
Buff
Cyan
Magenta
Orange
Black
NUM. MODE
RUN
CLEAR
AUTO INSERT
COLOR
MAGENTA LOWER CASE
ORANGE AUTO INSERT
BLUE NUMERIC
BUFF CLEAR KEY (TOP)
RED BREAK KEY (LFT)
platinum software

PLATINUM SOFTWARE WORKSAVER

FULL SCREEN EDITING

The following few pages discuss the techniques of using the FULL SCREEN EDITOR for enterin and editing BASIC programs. Our discussion is centered around the entry of a very simple full scree array editor. We will demonstrate the use of this array editor in the next section, USING THE INPU COMMANDS. In this section we will cover editing the BASIC line, relocating lines, auto line numberin joining lines, splitting lines, and finally dynamic editing. The dynamic discussion editing includes a example of dynamically clearing additional string space.

EDITING BASIC LINES

The initial line of our full screen array editor example is the data line shown below,

```
1 DATA  EGGS.MOLK,BREAD
```

The errors in this line are intentional. You should enter the line as shown and we will discuss how it i edited.

Do you know why a magenta block appeared between the 1 and the 'D' in 'DATA' as you were typin the line? It is there to remind you to press the ENTER KEY. The magenta block appears whenever eithe a new line is being entered or an existing line has been changed. Without the magenta block, it is eas to make changes to the screen and forget to 'ENTER' them. The magenta block will vanish when yo press the enter key, which you should have already done at this point.

Next list line 1 by (CLEAR), (L), (ENTER). Use the arrow keys to position the cursor over the '.' betwee the 'EGGS' and the 'MOLK'. Now type a ',' and the '.' is overwritten by the ','. Furthermore, a magenta block has again appeared between the '1' and 'D' to indicate a change has been made to this line o the screen. Finish correcting the line by changing the 'O' in 'MOLK' to 'I' and pressing the ENTER ke with the cursor flashing over the 'L' in 'MILK'.

The cursor will end up over the 'O' in 'OK', but you don't have to move the cursor because you ca just write over the 'OK'. For example, let's list line 1 once again, (CLEAR), (L), so that the 'OK' is over written by the command 'LIST'. Your screen should look like this:

```
1 DATA EGGS.MOLK,BREAD                    —   dashes represent the right border of the
                                                  screen
LIST                                      —
1 DATA EGGS,MILK,BREAD                    —
LIST                                      —
1 DATA EGGS,MILK,BREAD                    —
OK                                        —
```

As you can see, when you made the correction to line 1, all of line 1 was entered EVEN THOUGH THE CURSOR WAS OVER THE 'L' IN 'MILK' AND NOT AT THE END OF THE LINE. THE WORKSAVER KEEPS TRACK OF THE BEGINNING AND END OF LINES (see splitting lines section) SO THAT THE WHOLE LINE IS ENTERED WHENEVER THE ENTER KEY IS PRESSED.

Now move the cursor to the top line (the one with 'MOLK'), and press the ENTER key. The cursor will now be flashing over the 'L' in 'LIST'. Without moving the cursor, press the ENTER key again, and your screen should look like this:

```
1 DATA EGGS.MOLK,BREAD                    —
LIST                                      —
1 DATA EGGS.MOLK,BREAD                    —
K                                         —
DATA EGGS,MILK,BREAD                      —
```

THIS IS FULL SCREEN EDITING! MOVE THE CURSOR TO ANY LINE, HIT THE ENTER KEY AND WHATEVER IS ON THAT LINE IS ENTERED AS THOUGH YOU JUST TYPED IT. Now move the cursor to the bottom line (the one with 'MILK') and enter it so that the correct line is in the program.

RELOCATING/COPYING BASIC LINES

Lines are very easy to relocate using the WORKSAVER. Just change the line number and press the enter key. There are a few things to keep in mind when you do this, however, and these are discussed in the next two paragraphs. These paragraphs are not part of the FULL SCREEN EDITING EXAMPLE that we have been discussing so far. If you are following that example, you should not type anything until you reach the EXAMPLE again.

The first thing to keep in mind when changing line numbers is that you end up with two copies of the line (the one with old line number and the one with the new line number), and you may want to erase the original line. This can be done using the DEL command or by just typing the line number of the line you wish to erase and then pressing the enter key, i.e., (2), (ENTER) will delete line 2.

The second thing to keep in mind when relocating lines is that previous GOTO or GOSUB assignments to the old line (if you have deleted it) remain unchanged and will cause an undefined line error.

HINT: You can test for undefined lines by using the RENUM command as follows: Assume the largest line number in your BASIC program is 225, then entering RENUM 226,226,1 will list the line numbers that are undefined and the lines that they are called from. You can now use this information to update any GOTO or GOSUB calls that are wrong.

EXAMPLE: Back to the array editor, change the line number on the line with 'MILK' to line 5 and press the ENTER key. Now clear the screen, (SHIFT) + (CLEAR), and LIST the program by (CLEAR) , (L). Your screen should look like this:

```
LIST
1 DATA EGGS,MILK,BREAD
5 DATA EGGS,MILK,BREAD
```

Now delete line 1 by moving the cursor up to line 1 and placing it between the 1 and 'D'. Next delete everything to the right of the cursor and press the enter key. The key sequence is

(CLEAR) , (RIGHT ARROW) , (ENTER)

Line 1 is now deleted and the cursor is flashing over the 5 on line 5.

AUTOMATIC LINE NUMBERING

This is a common feature of most advanced BASICS which allows the user to specify a starting line number and increment. The computer then prints the line numbers for you as you enter your program. That is the way most AUTO LINE NUMBERING modes operate, but the WORKSAVER adds flexibility by being a little different as explained below.

The information given below is not part of the FULL SCREEN EDITING EXAMPLE, and if you are following the example, you should not type anything until you reach the EXAMPLE again.

To invoke the auto line numbering mode, press

(CLEAR) , (0)

and the symbol,

#

is printed. Now enter ONLY the increment (not both starting line number and increment) to be used in numbering your lines, i.e.,

#2

would set the increment to 2.

To use the auto line numbering mode, you enter the first line exactly as you normally w number and all. Then after it is entered, the WORKSAVER adds the increment to its line nu automatically prints the next line number for you. Normally you would then enter the code fo gram line, but the WORKSAVER also allows you a few other options. You can back the curso the line number is printed and then type over the number with either a different number or mand such as LIST. If you type over the number with a different number then after the ne entered the WORKSAVER will again add the increment to that number and print the next su line number. This also allows for editing lines which are still on the screen but were enter AUTO LINE NUMBERING MODE (ALNM). If you make any changes to a line on the screen and ENTER key, that line will be reentered and the ALNM will print the next successive line nur

The ALNM is 'temporarily' disabled by,
(BREAK) , (0)
which erases the line number. The ALNM will then automatically reappear whenever anoth either edited or entered. At which time, the increment that you were using when you disa ALNM will be added to that line number and the next successive line number will be printe WORKSAVER.

To turn off the ALNM requires setting the increment to 0 by the following sequence;
(CLEAR) , (0) , (0) , (ENTER).

HINT: The ALNM can be used to renumber a block of lines by listing the lines such that line to be moved remains on the screen. Now change the line number of the first press the ENTER key. The next line will be renumbered by ALNM, and pressing th key will copy and move it. WARNING: WHEN THE ALNM REWRITES LINE NUMBER LY ATTEMPTS TO WRITE OVER THE PREVIOUS LINE NUMBER. IT WILL THE LEAVE PART OF THE OLD LINE NUMBER IF THE NEW LINE NUMBER HAS LES THAN THE OLD LINE NUMBER, e.g., ALNM WILL OVER WRITE LINE NUMBER 2 LINE NUMBER 12, BUT THE RESULT WILL BE 125 NOT 12. FORTUNATELY, THE IS CLOSE BY SO IT IS EASY TO DELETE OUT THE 5 AND LEAVE THE 12.

EXAMPLE: Back to the FULL SCREEN EDITOR EXAMPLE. We will enter the next 3 lines of the using the ALNM with an increment set at 5. To set the increment to 5, use the key sequen below,
(CLEAR) , (0) , (5) , (ENTER),

and type in the next lines of the array editor as shown below;

```
10 FOR  N=1  TO  3
15 READ  A$(N)
20 NEXT
```

You will type the 10 for line 10 but the WORKSAVER will print line numbers 15, 20, and 25 for do not have a line 25, so at this point we will temporarily disable the ALNM by
(BREAK) , (0)
which erases the 25.

LINE JOINING

Now we will join these three lines together. First move the cursor to the '1' of '15' in line nu and press the LEFT ARROW once to move the cursor to the right side of the screen on the sam line as line 10. The cursor will be flashing white. This is the end-of-line-character (EOLC for s line 10. Delete this character by (SHIFT) + (LEFT ARROW), but, to stay consistent with this exa careful NOT to hold the left arrow down more than 1/2 second or you will delete the '1' in 15 als

the EOLC is deleted, the 1 in the 15 is pulled around to where the cursor is, and your screen s
like this;

```
10 FOR N=1 TO 3                    1—    cursor over the 1
 5 READ A$(N)                       —
20 NEXT                             —
```

Now you can delete the 1 and the 5 by holding down the (SHIFT) and (LEFT ARROW), but be c
to erase the 'R' in 'READ'. After the 1 and 5 have been erased, do a back search for the '3' in li
is done by the following key sequence;
(@) , (LEFT ARROW) , (3)
which will put the cursor over the 3 in line 10. Now type a colon after the 3, and delete out t
between the ':' and 'READ' by again (SHIFT) + (LEFT ARROW).

You have now joined lines 10 and 15, but line 15 still exists as part of your program. We ca
ALNM to delete line 15! When the ENTER key is pressed the newly formed line 10 is entere
ALNM is reactivated, resulting in the number 15 being automatically printed. Now line 1
deleted by simply pressing the enter key.

At this point, we will skip repeating this procedure for joining line 20 to line 10, and sho
easier method which utilizes the REDEFINE KEY features of the WORKSAVER.

To see how this is done, first clear the screen and enter the original lines 10, 15 and 20 ag
the cursor to the 'R' in line 15 and press (BREAK) , (SHIFT) + (@). A red checkered block will be
before the 'R'. Now move the cursor to the right of the ')' in 'A$(N)' (try using (@) , (SPACE BAR
(:) after the ')' and press (SHIFT) + (@) , (RIGHT ARROW). The block, 'READ A$(N):', is now st
you will be able to use the ERASE LAST KEY feature, (BREAK) , (ENTER), to erase the block
screen. After you delete 'READ A$(N)', you can delete line 15 from the program by pressing th
key.
Of course to delete line 15 using the ENTER key, we assume your cursor is next to the 15 t
on the screen after you deleted 'READ A$(N)'.

The cursor is now on line 20 flashing over the 'N' in 'NEXT' as a result of the ALNM printi
again. Next put the WORKSAVER in AUTO INSERT MODE by (SHIFT) + (SPACE BAR) (remer
space bar is an auto repeat key so be careful not to hold it down too long). Now press (BREAK)
ARROW) to print the contents of line 15 to the screen, and thereby join it to line 20. Your scree
now look like this;

```
10 FOR N=1 TO 3                    —
15                                  —
20 READ A$(N):NEXT                  —
```

with the cursor flashing over the 'N' in 'NEXT'. Now move the cursor to the 'R' (try using (@) , (L
ROW) , (R)), and use block get to store 'READ A$(N):NEXT'. This is done by
1. (BREAK) , (SHIFT) + (@);
2. moving the cursor to the right of the 'T' in 'NEXT';
3. (SHIFT) + (@) , (RIGHT ARROW).

Next erase the block by (BREAK) , (ENTER) and delete line 20 by (ENTER).

Now to join the saved line to line 10, move the cursor to the right of the '3' in line 10 and type
The block can now be PUT by (BREAK) , (RIGHT ARROW). The final result will look like

```
10 FOR N=1 TO 3:READ A$(N):NEXT    —
15                                  —
20                                  —
```

SPLITTING LINES

Now that we can join lines, it is time to learn how to separate them. But first we need to discuss ho the WORKSAVER defines lines. To do this type (CLEAR) , (UP ARROW) and all the end of line positio currently defined are represented by an inverted right bracket. A line is defined as everything on t screen between two end of line characters (spaces between the last character of a line and the end line character are ignored).

You've probably already guessed that to split a line, an end of line character is inserted where t break is to occur. O.K. let's try it. Move the cursor to the ':' between the '3' and 'READ'. Next pre (CLEAR) , (DOWN ARROW) and an end of line character is printed over the colon. Furthermore, t WORKSAVER is now in the AUTOMATIC INSERT MODE in anticipation of your needing to insert a li number. Go ahead and enter (15) , (SPACE BAR) , (ENTER). Your screen should look like:

```
10 FOR N=1 TO 3#15 READ A$(N):NE   —      where the symbol '#'
XT                                 #—       represents the end of line
20                                 #—       character
20                                 #—
```

At this point line 15 is in your program, but the shortened version of line 10 has not been entered indicated by the magenta block on 10. Move the cursor up to line 10 and press (ENTER). The new defined line 15 is erased from the screen when this is done, but don't be alarmed, it's still part of yo program and you can prove this by listing the program, but to stay consistent with this text you shou complete this section on line splitting before you list the program again.

WARNING: The ALNM has printed the 15 for line 15 and if you press the ENTER key now, you will era line 15.

To avoid erasing line 15 we will now turn off the ALNM by:

1. (BREAK) , (0) : erases the 15
2. (CLEAR) , (0) , (0) , (ENTER) : sets increment to 0 which totally disables the ALNM.

Now turn off the end of line characters by (CLEAR) , (UP ARROW). This changes them to solid gre graphics character, HEX8F, which is why they cannot be seen except when the cursor flashes ov them. It is not necessary to turn them on in order to split the line. We only did it here to make the poi about how line splitting worked.

One final note: THE SPACES BETWEEN THE END OF LINE AND THE LAST PRINTED CHARACTE IN A LINE (there were 29 spaces between the 'T' in 'NEXT and the end of line character for line 15 the screen shown above) ARE NOT TAKEN AS PART OF THE LINE WHEN THE LINE IS ENTERED. FOR SOME AESTHETIC REASON, YOU WANT SPACES AFTER A LINE, USE THE EXTENDED BAS LINE EDITOR.

DYNAMIC EDITING

Dynamic editing is our term for making corrections to programs while running a program. This ca not be done with MICROSOFT BASIC because changes to programs automatically resets all variable arrays, and strings. The program must then be restarted and all tape loads and data entries must b redone. The WORKSAVER'S DYNAMIC EDITING reduces the need to reload and reenter data each tim a change is made to your program. We will demonstrate how it works by continuing with our simp program of a FULL SCREEN ARRAY EDITOR.

Let's begin with a clean slate. Clear the screen and list the program as it stands now. The progra should look like:

```
5 DATA EGGS,MILK,BREAD      —
10 FOR N=1 TO 3             —
15 READ A$(N):NEXT          —
```

To demonstrate DYNAMIC EDITING we will create an error, and we can do this by changing t between 'A$(N)' and 'NEXT' to a '—'. After this change is made, execute the program by

(SHIFT) + (ENTER)

Now part of your screen should look like,

```
5 DATA EGGS,MILK,BREAD                    —
10 FOR N = 1 TO 3                         —
15 READ A$(N)—NEXT                        —
?SN ERROR IN 15                           —
OK                                        —
```

The SYNTAX ERROR is corrected by changing the '—' in line 15 to a ':' and pressing the ENTER Of course the DYNAMIC EDITOR is totally transparent (except for a slight delay that occurs when have a lot of data). The theory of operation of the DYNAMIC EDITOR was discussed in C.MEM, so just show the results here.

At the time of the SYNTAX ERROR, N and A$(1) were 1 and 'EGGS' respectively, and now that have edited line 15, they should still be the same. This can be demonstrated by printing N and A$(follows:

```
?N;A$(1)
```

To reenter the running of your program use GOTO or GOSUB. If you use the RUN command, variables, arrays, and strings are all reset and the advantages of DYNAMIC EDITING are lost.

If you have read ahead, you are aware that there are limitations to the use of DYNAMIC EDITI When a program is structured with these limitations in mind, they will be minor. Even without careful consideration of these limitations in the structure of a program, the major advantage DYNAMIC EDITING will generally not be effected. That advantage is to avoid having to reloa from tape each time a change is made. Unfortunately, there will be times when you will have to re your data; however, you should at least be able to use the data as it exists to test out whatever cha you've made before reloading.

Now we will present examples that exemplify the limitations listed at the end of this section. At point, line 20 has been corrected, and we are ready to reenter the program. The first WARNING tells not to reenter the program inside a FOR/NEXT loop; consequently, we will reenter this program at 10. So go ahead and enter

(BREAK) , (G) , (10) , (ENTER)

You now have an '?OD ERROR IN 15' message. This occurred because at the time of the syntax e above the first data element, EGGS, had been read. When you reentered the program at line 15, line attempted to read the next 3 data elements, but there were only 2 left, MILK and BREAD. Thus whe program is suspended the internal data is not RESTORED, and you will have to decide if it should restored before you reenter the program. In this case it should be RESTORED, and this is done b

(BREAK) , (T) , (ENTER)

A (GOTO 10) will now run without any errors. But before you execute the GOTO, add lines 16 and 17 given below

```
16 'A STRING INSIDE A PROGRAM
17 A$ = "STRING"
```

Now run the program by (GOTO 10). You should not have encountered any errors. We can now sho
you why string assignments inside a program (like line 17) may cause problems. First Print A$ by (?A$
Now delete line 16 by (16), (ENTER). This effectively moves the location of line 17 and thus the locatio
of A$ in memory, but BASIC will look for A$ at the old location. Check the result by again printing A$
You'll notice A$ has been changed. Now change line 17 to look like

```
17 A$ = "STRING" + ",
```

and move the cursor back up to where line 16 was orginally entered and reenter it (move cursor to lin
16 and press ENTER key will reenter line 16). Run the program by (SHIFT) + (ENTER).

Now print A$, delete line 16, and print A$ again. This time A$ was not affected by deleting line 1

String data, as specified in line 5, is also located within the program. As such, you cannot relocat
those lines either when using dynamic editing. For example, add line 2 as given below

```
2 'THIS REMARK WILL CHANGE A$(1)
A$(2),A$(3)
```

Now print A$(1) and you will find that it has changed.

TURNING OFF DYNAMIC EDITING

There will be times when you will want to reset your data. You do this by '?1:CLEARn' when n is a
number. Basic resets all variables, arrays, and strings when a clear command is executed. T
Worksaver does not invoke dynamic editing when the CLEAR command is not the first command or
line entered from the keyboard.

Hopefully you now have a better understanding of DYNAMIC EDITING and its limitations. The limi
tions are listed below.

WARNING: When using GOTO and GOSUB to reenter your program, avoid entering your program in t
middle of a FOR/NEXT loop. If you do reenter inside a loop you will get a '?NF ERROR'.

WARNING: When you reenter your program after DYNAMIC EDITING you will have to decide if y
need to restore the internal data.

WARNING: String data, such as

```
9 DATA EGGS,MILK,BREAD,
```

and String assignments inside a basic program, such as

```
10 A$ = "STRING"
```

can be changed by DYNAMIC EDITING. This can happen because the strings are located inside
program and increasing or decreasing lines 0 thru 8 will change the location of lines 9 and 10
memory. Consequently basic can no longer find the strings at the same location.

This problem can be avoided by adding the NULL STRING, " ", to any string assignment made ins
a basic program. Thus changing line 10 to the following,

```
10 A$ = "STRING" + " "
```

will solve the problem by moving the location of A$ from inside the basic program to the area
memory called STRING SPACE where it is protected by DYNAMIC EDITING.

Likewise, adding the null string to the read statement as follows will also move the data string to
ing space.

```
11 READ  A$(N):A$(N) = A$(N) + ""
```

The consequence of using the null string, is that you use twice as much memory to store the str
You end up with a copy of the string in the program and one in string space. For this reason, we s
gest that you avoid the need for null strings by locating your string assignments and data stateme
at the beginning of programs so that it is unlikely that they will be affected by any editing you may n
to do while running the program.

STRING SPACE AND DYNAMIC EDITING

Back in the FUNCTIONS SECTION under the discussion of C.MEM, we promised an exam[illegible] ing additional string space using DYNAMIC editing.

We will begin the example by entering line 16 as follows:

```
16 FOR X = 1 TO 10:A$(1) = A$(1) + A$(1):NEXT
```

Next delete line 17, clear the screen, and list the program. It should look like

```
5 DATA EGGS,MILK,BREAD
10 FOR N = 1 TO 3
15 READ A$(N):NEXT
16 FOR X = 1 TO 10:A$(1) = A$(1) + A$(1):
NEXT
```

The program can now be run, and it will generate an ?OS ERROR IN 16 (as long as the string sp[illegible] still 100 bytes as it is when the WORKSAVER is initialized). To check string space reserved (BREAK) , (B) and you should get the following numbers

```
100 64 n n
```

where the 100 is the amount of string space currently reserved, the 64 is the amount used, and th[illegible] n's represent the start of machine language space minus one and the start of the WORKSAVER m[illegible] one respectively.

We will expand the string space another 100 bytes by
(CLEAR) , (B) , (300).
We can now expand string space 100 bytes by (CLEAR),(B),(200),(ENTER) and the A$ array will n[illegible] lost. You can demonstrate this by ?A$(2).
We have now increased string space by a total of 200 bytes without resetting any data, and you test this by printing A$(2).

USING BASICS INPUT COMMANDS

The WORKSAVER is designed to be used as a full screen editor even when you run a BASI GRAM. The real demonstration of this is given with the FULLSCREEN ARRAY EDITOR which is flip side of the WORKSAVER tape. In this section we will introduce you to the fundamentals of c ing a full screen array program.

By now you should be able to do full screen edit BASIC programs so we will leave it up to finish editing and entering the program as shown below:

```
5 DATA EGGS,MILK,BREAD
10 CLS:FOR N = 1 TO 3
15 READ A$(N)
20 N$ = RIGHT$("          " + STR$(N),3)
25 PRINT"N,ELEMENT?" + CHR$(143) + N$ + "," + A$(N)
30 NEXT:N = N—1
35 PRINT@N*32,"";:INPUT"N,ELEMENT";N,A$
40 A$(N) = A$
45 GOTO 35
```

You should be familiar with lines 5 thru 15, so we will begin our discussion with line 20. This forms the same function as PRINT USING "###";N would in EXTENDED basic. It converts the N to a right justified string of length 3 that contains N. The resulting N$ is what we refer to as th TROL STRING'.

Line 25 sets up the screen for full screen editing. Each element of the A$ array is print separate line. Each line contains a 'N,ELEMENT?' string. This string is there to simulate the q that is printed by the BASIC command INPUT. After the question string we have printed the g character 143 which is a solid green block. When this block is printed, it will look just like the green background, but the WORKSAVER will see it as a marker on the screen. It is the end marker that was discussed in the LINE SPLITTING section. In full screen editing the marker se the INPUT question from the rest of the line as discussed later. After the marker, we print the string, N$; a ','; and the array element, A$(N). This combination simulates the entry of mix (numeric and string) for an INPUT command. When line 25 prints all three array elements, the will look as if you just entered them as answers to an INPUT prompt.

The N = N-1 in line 30 is there to set N = 3, the last array element printed, after the FOR/NE This setups the location used in line 35 for the next screen entry. When line 35 is reached, thr will have been printed to the screen. Each line is 32 characters long. Thus, the next line b screen location 3*32.

Line 35 uses the value of N to position the INPUT question on the screen at a logical point PUT command will print the question, 'N,ELEMENT', and follow it with a '?'. The WORKSAVER AUTOMATICALLY print the end of line marker, graphics character 143, that separates the q from the data input on the screen. THIS MAKES THE FULL SCREEN EDITING FEATURE WORKSAVER COMPATIBLE WITH THE INPUT PROMPTS OF ANY COLOR COMPUTER BAS GRAM.

The answer to the INPUT prompt in line 35 will be the number of the array element to be en well as the array string itself. You can therefore enter any element you wish by specifying its and with the fullscreen, you can move up and down the screen changing and reentering ele will.

NO MORE EDIT MENUS!!!!

Line 40 takes the N and the A$ from the input in line 35 and updates array element A$(N).

Line 45 puts the program in an infinite INPUT loop. We will use this loop to demonstrate how to int rupt the program when it is waiting for an answer to an INPUT command.

RUNNING THE PROGRAM:

Now run the program (SHIFT) + (ENTER).
If you haven't made any errors in the program, your screen should look like:

```
N,ELEMENT? 1,EGGS
N.ELEMENT? 2,MILK
N,ELEMENT? 3,BREAD
N,ELEMENT?
```

If you have an error; well at least now you can make an easy correction.
Assuming you now have a working program, go ahead and use it to add to the array and ma changes anywhere on the screen.

INTERRUPTING BASIC INPUT COMMANDS

When you want to interrupt the program, use (SHIFT) + (BREAK), and the program will be halte (SHIFT) + (BREAK) will interrupt all programs when they are waiting for input. Programs that are e ecuting commands other than an input (i.e. loops, subroutines, INKEY$, etc) are interrupted by t BREAK key without the SHIFT.
When you are ready to move on to the next section, PROGRAM CHAINING, you should leave this pr gram in an INTERRUPTED state. We will use it to demonstrate PROGRAM CHAINING.

PROGRAM CHAINING

Program chaining will allow you to run a program that is too large to be loaded as one program. Th is done by splitting the program into unique parts and saving each part as a self contained progra You then load and run each part one at a time without having to save and reload your data each tim you load a new part. With the WORKSAVER all you do is load the next program part and all the da generated by the last part is still in memory.
Program chaining is automatic. Whenever a CLOAD command is executed (not as part of a program the data that existed at that time is first protected and then attached to the new program part when has completed loading.
Before we demonstrate program chaining, you might want to save the example program you entere in the last section. You can do this by CSAVE "EXAMPLE" ((BREAK) , (9)).
Now we will demonstrate program chaining by using the data from the program you interrupted i the last section. When you interrupted that program you had an A$ array in memory, and to prove that is still in memory, you should pick one of the elements, say A$(3), and print it by ?A$(3).
When you are ready, load the FULL SCREEN ARRAY EDITOR that we included with th WORKSAVER. The ARRAY EDITOR is saved at the beginning of the flipside of the WORKSAVE cassette. You can load it by (CLEAR) , (9) , (ENTER). When the program is finished loading, print A$(3 again. It is still in memory along with all the rest of the data that was generated by the example pro gram in the last section.
The ARRAY EDITOR you just loaded will not use any of this data as it would normally be used if w were really chaining the two programs. Despite this difference, we will present the final step requ to complete the chain. This step involves executing the program you just loaded without losing th data you have protected to this point. A RUN command will wipeout the data, so to link the program you must execute a GOTO command, and the GOTO command should GOTO the first line to be ex ecuted.
Now you can turn to the next section and run a multi featured array editor.

PLATINUM SOFTWARE WORKSAVER

FULL SCREEN ARRAY EDITOR

The FULL SCREEN ARRAY EDITOR that you have just loaded in the PROGRAM CHAININ is excellent for maintaining a list of items that are to be frequently updated. We have demonstrate it as a shopping list, but it can be used for other lists as well.

The program is only a start. It contains the routines for updating the array and printi screen. With this program you can:

1) change any item in the array that is on the screen by typing over it and pressing th
2) insert a new element between any two elements in the array,
3) scroll the array up and down the screen, and
4) move individual elements of the array from any location to any other location.

The program has plenty of room for improvement, and you'll probably want to add at least following ideas:

1) a tape load and tape save routine (we have put these choices in the menu but t routines that do them),
2) a print the array to printer routine (this is also in the menu but there is no routin
3) capability to handle string elements longer than 26 characters,
4) capability to sort the array,
5) capability to handle numeric arrays, and
6) capability to handle multi-dimensional arrays and etc.

INSTRUCTIONS

To begin using the array editor you must first start it by pressing (SHIFT) + (ENTER. Aft RUN, your screen will look like

```
 1EGGS                    —
 2BACON                   —
 3BREAN                   —
 4STEAK                   —
 5SPAGHETTI               —
 6CHICKEN                 —
 7MILK                    —
 8ORANGE JUICE            —
 9LETTUCE                 —
10PAPER TOWLS             —
11LAUNDRY DETERGENT       —
12APPLES                  —
13POTATOES                —
14SOUP                    —
D-SCRDN U-SCUP M-MOVE     —
S-SVTAPE L-LDTAPE P-PRNTR —
```

As you can see we printed a grocery list where each item in the list is a separate elemen array. The screen contains array elements 1 thru 14 plus a menu of single letter comman

CHANGING ITEMS IN THE ARRAY:

You can now use the arrow keys to move the cursor to any element on the screen. To cha

just type over the item leaving the number of the item untouched. The item number resides in wh
will be referring to as the control field. The control field is the left 4 columns of the screen. Do
anything in this field when you are only changing an item on the screen.

As soon as you make a change to an item, a magenta block will appear in the control field. This
indicates that a change has been made on the screen but has not been entered into the array.
the enter key is pressed, the change on the screen is entered into the array. Thus, the magenta blo
a reminder to press the enter key after making changes on the screen.

EXAMPLE

Type FISH over MILK in item 7 and press the ENTER key. The magenta block is erased after pres
the ENTER key and the cursor is now in column one of the control field on the line with item 8.

INSERTING ELEMENTS

Let's assume you wish to add BUTTER to this list, and that you want it to be between items BR
and STEAK. You do this by entering any fractional number between 3 and 4, say 3.5, into the co
field and then typing in BUTTER in the array field. Furthermore, you can do this anywhere on the sc
below item 3, and the screen will be automatically updated.

EXAMPLE

Enter 3.5 and butter on the line containing item 8. Lines 7, 8, and 9 should now look like the follow

```
 7FISH
3.58BUTTER                    —
 9 L E T T U C E              —
```

We just changed MILK to FISH.
You can leave the 8 and type over the ORANGE JUICE with BUTTER - SPACE BAR or SHIFT + LEFT ARROW or CLEAR,RIGHT ARROW can be used to erase the word JUICE.

After the enter key is pressed the screen will look like:

```
 1EGGS                        —
 2BACON                       —
 3BREAD                       —
 4BUTTER                      —
 5STEAK                       —
 6SPAGHETTI                   —
 7CHICKEN                     —
 8FISH                        —
 9ORANGE JUICE                —
10LETTUCE                     —
11PAPER TOWLS                 —
12LAUNDRY DETERGENT           —
13APPLES                      —
14POTATOES                    —
D-SCRDN U-SCUP M-MOVE         —
S-SVTAPE L-LDTAPE P-PRNTR     —
```

As you can see, BUTTER is now element number 4. All elements greater than 4 have beer
renumbered, and the screen is updated to show this.

DELETING ITEMS

An element of the array is deleted by placing a minus sign in the control field of that element.

EXAMPLE

Suppose you wish to delete LAUNDRY DETERGENT. All you do is place a '-' in the control field as shown below)

```
  11PAPER TOWLS                  —
- 12LAUNDRY DETERGENT            —
  13APPLES                       —
```

After the ENTER key is pressed, the screen will be updated as partially shown below;

```
   9ORANGE JUICE                 —
  10LETTUCE                      —
  11PAPER TOWLS                  —
  12APPLES                       —
  13POTATOES                     —
  14SOUP                         —
D-SCRDN U-SCUP M-MOVE            —
S-SVTAPE L-LDTAPE P-PRNTR        —
```

Again the array is renumbered after deleting LAUNDRY DETERGENT.

MOVING ELEMENTS UP AND DOWN THE ARRAY

Elements can be moved from any location in the array to any other location. This is done by:

1. placing an M in the first column of the control field of the element,
2. pressing the ENTER key,
3. using the UP or DOWN ARROW KEYS to move the element.

After the element has been moved, press any other key to return to the array editor.

EXAMPLE

We will move ORANGE JUICE from between FISH and LETTUCE up the screen, and put it between BACON and BREAD. To do this, place an M in the first column of the control field of ORANGE JUICE as shown below.

```
   8FISH                         —
M  9ORANGE JUICE                 —
   10LETTUCE                     —
```

Now press the enter key and use the UP ARROW key to move ORANGE JUICE Up the list until it is between BACON and BREAD. Each time you press the UP ARROW KEY ORANGE JUICE moves one position up the list, and the screen is automatically updated to show this. When you have finished moving ORANGE JUICE your screen will look like this;

```
   1EGS                          —
   2BACON                        —
M  3ORANGE JUICE                 —
   4BREAD                        —
   5BUTTER                       —
   6STEAK                        —
   7SPAGHETTI                    —
   8CHICKEN                      —
```

```
 9FISH                                  —
 10LETTUCE                              —
 11PAPER TOWLS                          —
 12APPLES                               —
 13POTATOES                             —
 14SOUP                                 —
D-SCRDN U-SCUP M-MOVE                   —
S-SVTAPE L-LDTAPE P-PRNTR               —
```

The M stays with the element being moved until you return to the editor by pressing another key. You should now experiment with moving the EGGS from the top of the list to the bottom (beyond SOUP), and then move it back up again.

SCROLLING THE ARRAY

The array can be continuously scrolled by placing either an U or a D in the first column of the control field on any line. Scrolling the array is the way you get any element in the array to appear on the screen for editing. Once the array starts scrolling either up or down the screen, you stop it by pressing any key.

ADDING ELEMENTS TO THE ARRAY

The ARRAY EDITOR will place a blank element after the last element in the array. You can see this by scrolling the array either up or down, and then stopping it as explained above. To add to the array just enter another item in the blank area. The array editor will automatically print another blank line after it for the next item to be added.

EXAMPLE

Assume you used a D and scrolled the screen down, and then stopped it in the following position.

```
10LETTUCE
11PAPER TOWLS                           —
12APPLES                                —
13POTATOES                              —
14SOUP                                  —
15BROCCOLI                              —
16                                      —
 1EGGS                                  —
 2BACON                                 —
 3ORANGE JUICE                          —
 4BREAD                                 —
 5BUTTER                                —
 6STEAK                                 —
 7SPAGHETTI                             —
D-SCRDN U-SCUP M-MOVE                   —
S-SVTAPE L-LDTAPE P-PRNTR               —
```

The shopping list array only has 15 items. If you now enter another item at 16, the editor will place an empty item 17 below it.

APPENDIX 1
ARRAY EDITOR PROGRAM LISTING

The following is a commented listing of the fullscreen array editor. We hope that you will be able to dissect its operation and modify it for your own personal needs.

The variables used in this program are as follows;

- LT$( ) - The string array that is edited by the program
- MX - The dimension of LT$. The program prevents you from exceeding this number of elements.
- NX - The next available element of LT$. As data is entered into LT$ it always resides between element 1 and NX-1, i.e., there are no null strings in this range. When elements are deleted the array is reordered.
- A - The screen location used to print array elements to the screen. Most of the program deals with determining the value of A.
- N - The element number that is to be printed at location A.
- A$ - The string entered from the screen. A$ contains a 5 element control field and the rest is information that is to be placed in the array (the array field).
- X - The value of the control field in A$. This is the variable that tells the array editor what to do with the rest of A$.
- T - The array-element number that is printed at the top of the screen.
- SX - The value of NX since last screen scroll. SX is used by delete and insert.

LINES 2-10

Lines 2-10 enter a machine language program which ties into the WORKSAVER routine for scrolling down the screen. The assembly listing is as follows;

BDB3ED	JSR $B3ED	This is the INTCNV routine listed in RADIO SHACK'S EXTENDED BASIC MANUAL ON PAGE 147. The USR function in our program below has the screen location as its argument, and the INTCNV converts it to an integer in the D register.
3406	PSHS D	The WORKSAVER expects the screen location to be in the stack.
8E05C0	LDX #$5C0	The WORKSAVER will scroll the screen down to this location (the screen is located between &H400 and (H5FF). Thus we have protected the bottom 2 lines of the screen from being scrolled.
7E@@@@	JMP SCRLDN	This is the WORKSAVER subroutine that scrolls the screen down. A LBRA to this subrouting is located at the beginning of the WORKSAVER. In the program below the beginning is calculated from address &H74 which points to the top of available memory. &H74 will also point to the start of the WORKSAVER when the DEFAULT is used in relocating the WORKSAVER after it is loaded.

```
2 GOSUB8:CLEAR2000,X:GOSUB8
5 DATA &HBD,&HB3,&HED,&H34,&H06,&H8E,&H05,&HC0,&H7E
6 A$ = RIGHT$)"0000" + HEX$(X + 13),4)
7 FOR N = X TO X + 8:READ A:POKE N,A:NEXT:
EFUSR0 = X:
X = VAL("&H" + LEFT$(A$,2)):POKE N,A:
A = VAL("&H" + RIGHT$(A$,2)):POKE N + 1,A:GOTO10
8 X = (PEEK(&H74)*256 + PEEK(&H75))-11:RETURN
```

APPENDIX 1 : PROGRAM LISTING

LINES 10-14

Lines 10 thru 14 initialize the array and the variables. These lines should be studied for adding load to the program.

```
10 CLS:DIM LT$(30):MX = 30
12DATA EGGS,BACON,BREAD,STEAK,SPAGHETTI,CHICKEN,MILK,ORANGE JUICE,LETTUCE,P
TOWLS,LAUNDRY DETERGENT,APPLES,POTATOES,SOUP,BROCCOLI
14 FORN = 1TO15:READ LT$(N):A = N*32-32:GOSUB90:NEXT:
NX = N:SX = N:N = 14:GOSUB 134:A = 32:T = 1:N = 2:GOTO54
```

LINE 54

Line 54 is the keyboard entry line. This line sets up the location for the next screen input. This is the logical location since the cursor can be moved anywhere else from here. The LINEINPUT comm prints an end of line character after the message is printed. In this case the messages is the null s so the end of line character is printed at the @ location. The @ location is either A-1 (the rightsi the line above) or it is A when A = O.

```
54 PRINT@((A-1)-(A = 0)),"";:LINEINPUT"";A$
```

LINES 60-80

Lines 60-80 determine the value of the control field.

```
60 X = VAL(LEFT$(A$,5))
```
(control field is the 5 left most character in A$.

```
62 IF X> = 1 AND X = INT(X) THEN 100
```
(goto 100 for a simple change to array element X

```
64 IF X<0 THEN 110
```
(X has a fractional component, such as 2.1, therefore insert this entry between elements 2 and 3.

```
66 IF X>0 THEN 120
```
(Negative X means delete this array element.

If none of the conditions in lines 62, 64, or 66 are met, then X = 0 and the control field is checked fo control letter, i.e., 'UDMPSL'.

```
80 X = 1 + INSTR("UDMPSL",LEFT$(A$,1)):
ON X GOTO54,130,140,150,54,54,54
```

We have left it up to you to write the PRINT to printer, SAVE, and LOAD routines.

LINE 90

This line prints array element N to the screen at location A.

```
90 PRINT@A,"";:PRINTUSING"####",N,:PRINTLT$(N):RETURN
```

LINES 100-103

These lines perform the function of adding another element to the array or either updating or replacin an existing element in the array.

```
100 IF NX = MX AND X  MX THEN PRINT @448, "*** array is full":GOTO 54
101 IF X  = NX THEN N = NX = :LT$(N-1) = MID$(A$,5):D = 0:GOTO113
102 LT$(X) = MID$(A$,5)
```

```
103 A = (PEEK(&H88)*256 + PEEK(&H89))-&H400
IF
(A = 32*(1 + X-T) OR A = 32*(X + NX-T))
AND A < = 416 THEN 54 ELSE 113
```

LINES 110-112

These lines insert the array element into the array.

```
110 IF NX = MX THEN X = MX + 1:GOTO 100
111 N = INT(X) + 1:FOR U = NX + 1 TO N + 1 STEP-1:LT$(U) = LT$(U-1):D = 1:NEXT
112 LT$(N) = MID$(A$,5)
```

LINES 113-117

These lines update the screen when an element is inserted, deleted or added to the array.

```
113 B = (N-T):C = (N + SX-T):A = 32*((B)0)) + C*-(C(15)):NX = NX + D
114 IF A<0 GOSUB142:GOTO 113
115 IF A>416 GOSUB132:GOTO 113
116 X = A:N = N-1:FOR A = A TO 416 STEP 32:N = N + 1:IF N = NX + 1 THEN N = 1:SX = NX
117 GOSUB90:NEXT:A = X-32*(X = 0):GOTO54
```

LINES 120-121

These lines reorder the array when an element is deleted

```
120 N = ABS(X):FOR U = N TO NX:LT$(U) = LT$(U + 1):NEXT:D = -1
121 GOTO 113
```

LINES 130-134

These lines scroll the screen up

```
130 A$ = INKEY$:IF A$ = ""GOSUB132:GOTO130
131 SX = NX:GOTO54
132 PRINT@480,"":T = T + $ THEN T = 1
133 N = T + 13:IF N > NX THEN N = N-NX
134 A = 416:GOSUB90:PRINT@448,"D-SCDWN U-SCUP M-MOVE
SVTAPE L-LDTAPE P-PRNTR";:RETURN
```

LINES 140-143

These lines scroll the screen down

```
140 A$ = INKEY$:IF A$ = ""GOSUB142:GOTO140
141 SX = NX:GOTO54
142 U = USRO(&H3FF):T = T-1:IF T = 0 THEN T = NX
```

To use the WORKSAVER'S scroll down routine, you must supply the location of the beginning of line minus one. In this program we wish to scroll the lines from the top of the screen down. The edge of the top of the screen is at location &H400 (1024) thus location &H3FF (1023) = &H4 (1023 = 1024-1).

```
143 A = 0:N = T:GOSUB90:RETURN
```

LINES 150-159

These lines move an element up or down the array and update the screen as it is moved.

```
150 N = VAL(MID$(A$,2,3)):A = (N-T)*32:IF A<0 THEN A = (N + NX-T)*32
151 B$ = INKEY$:IFB$ = ""THEN151
152 X = 1 + INSTR(CHR$(10) + CHR$(94),B$):ON X GOTO159,153,156
```

where chr$(10) and chr$(94) are the ASCII characters for the down and up arrows respectively.

move element down the array

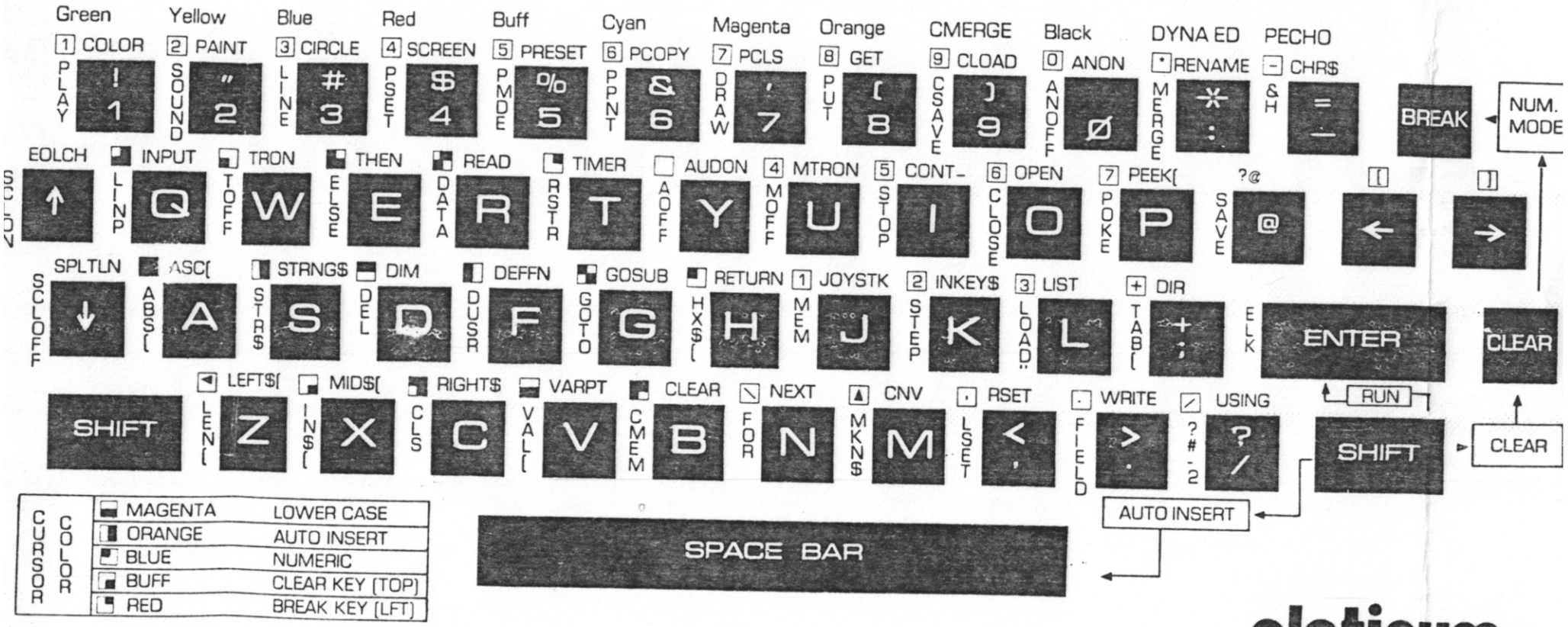
```
153 IF N + 1 = NX THEN 151 ELSE LT$(N) = LT$(N + 1):GOSUB90
154 N = N + 1:LT$(N) = MID$(A$,5):A = A + 32:IF A   416 GOSUB 132 ELSE GOSUB90
155 PRINT@A,LEFT$(A$,1),:GOTO151
```

move element up the array

```
156 IF N-1 = 0 THEN 151 ELSE LT$(N) = LT$(N-1):GOSUB90
157 N = N-1:LT$(N) = MID$(A$,5):A = A-32:IF A<0 GOSUB 142 ELSE GOSUB90
158 GOTO155
```

erase the 'M' and return to screen input

```
159 PRINT@A,"";:GOTO54
```



platinum software

WORKSAVER FEATURES

EDIT CONTROLS

Delete characters ..... (SHIFT) + (LEFT ARROW)
Erase to end of line ..... (CLEAR), (RIGHT ARROW)
Erase line ..... (BREAK), (0)
Insert spaces ..... (SHIFT) + (RIGHT ARROW)
Move cursor to left edge of screen . (CLEAR), (LEFT ARROW)
Split lines ..... (CLEAR), (DOWN ARROW)

SYSTEM CONTROLS

(CLEAR), (SHIFT) + (<) load table and append to existing BASIC program
(CLEAR), (SHIFT) + (>) load BASIC program and attach existing table to it.
(CLEAR), (SHIFT) + (9) cassette merge
(CLEAR), (SHIFT) + (*) dynamic edit: auto dynamic edit commands NEW, LOAD, CLOAD, CLEAR, PCLEAR, DEL, RENUMBER, MERGE

PROGRAM CONTROL FROM INPUT/LINEINPUT COMMANDS

DYNAMIC INPUT (SHIFT) + (ENTER); use '?' to compute total
PROGRAM BREAK (SHIFT) + (BREAK)

REDEFINE KEY PROCEDURES

((CLEAR) or (BREAK)), (SHIFT) + (@) initiate redefine key
1. enter new key definitions
2a. (SHIFT) + (@) defines print only type
2b. (ENTER) auto execute type
2c. (SHIFT) + (ENTER) transparent type
3. PRESS KEY to store new definition

@ KEY

(@), (ANY CHARACTER) search for character to right of key
(@), (RIGHT ARROW) move cursor up one line
(@), (LEFT ARROW) move cursor down one line
(@), (BREAK) move cursor to top left corner
(@), (CLEAR) move cursor to bottom left corner

PLUS FEATURES

PROGRAM LISTING CONTROL

*(SHIFT) + (UP ARROW) scroll program listing up the screen
*(CLEAR), (SHIFT) + (UP ARROW) delete spaces after line numbers from bottom of screen. (UP ARROW) deletes successive lines up the screen. Any key ends this function
*(SHIFT) + (DOWN ARROW) scroll program listing down screen
*(CLEAR), (SHIFT) + (DOWN ARROW) delete spaces after line numbers from top of screen. (DOWN ARROW) deletes successive lines down the screen. Any key ends this function

GLOBAL SEARCH AND REPLACE (SHIFT) + (@)

/.NX.	Simple search for NX throughout a program
/.NX.NEXTX.	Simple search for NX and replace with NEXTX throughout a program
/.NX.NEXTX.-100	/.NX.NEXTX.100-200 /.NX.NEXTX.-200 search and replace syntax for line ranges
/*.MAY. JUNE.	Search and Replace inside strings.
/'.SORT DATE.	Search remarks for SORT DATE
/. ..	Delete spaces in a program
/.'.	Search remarks in a program. USE .'. SYNTAX ONLY. Delete remarks by using @-delay and (CLEAR), (RIGHT ARROW).

LINE RENUMBERING

54;50	Renumber line 54 to be line 50
100-112;400	Renumber lines 100 through and including line 112 to be lines 400 through 412.
100-112;400,5	Renumber lines with an increment of 5

LINE RENUMBERING ERROR MESSAGES

?UL ERROR generated when the lines specified to be moved do not exist.
?FC ERROR generated when lines to be moved would overwrite or surround existing lines.
UL xxxx IN yyyy generated during renumbering process to indicate lines which are referenced but do not exist.

SCREEN SCROLL CONTROL

(BREAK), (UP ARROW) prints the # promt for number of lines to list between screen pauses.
(BREAK), (DOWN ARROW) cancels scroll control
(ANY KEY) continues listing or use (DOWN ARROW) to cancel control

PRINTER ECHO

(CLEAR), (SHIFT) + (=) echoes screen print to the printer. To turn off execute another (CLEAR), (SHIFT) + (=).
(CLEAR), (SHIFT) + (RIGHT ARROW) sends a logical screen line to printer. Subsequent (RIGHT ARROW)'s list next line to printer. Any other key ends this feature.