3 THE EDITOR (AND STAFF)

Up till now you've been compiling new definitions into the dictionary by typing them at your terminal. This chapter introduces an alternate method, using disk storage.

Let's begin with some observations that specifically concern the dictionary.

Another Look at the Dictionary

If you've been experimenting at a real live terminal, you may have discovered some things we haven't mentioned yet. In any case, it's time to mention them.

> Discovery One: You can define the same word more than once in different ways—only the most recent definition will be executed.

For example, if you have entered:

```
    : GREET   ." HELLO.  I SPEAK FORTH. " ; ok
```

then you should get this result:

```
    GREET HELLO.  I SPEAK FORTH. ok
```

and if you redefine:

```
    : GREET   ." HI THERE! " ; ok
```

you get the most recent definition:

```
    GREET HI THERE! ok
```

Has the first GREET been erased? No, it's still there, but the most recent GREET is executed because of the search order. The text interpreter always starts at the "back of the dictionary" where the most recent entry is. The definition he fir[illegible] fir[illegible] is the one you defined last. This is the one he shows to EXECUTE.

We can prove that the old GREET is still there. Try this:

```
    FORGET GREET ok
```

and

```
    GREET HELLO.  I SPEAK FORTH. ok
```

(the old GREET again!)

The word FORGET looks up the given word in the dictionary and, in effect, removes it from the dictionary along with anything you may have defined since that word. FORGET, like the interpreter, searches starting from the back; he only removes the most recently defined version of the word (along with any words that follow). So now when you type GREET at the terminal, the interpreter finds the original GREET.

FORGET is a good word to know; he helps you to weed out your dictionary so it won't overflow. (The dictionary takes up memory space, so as with any other use of memory, you want to conserve it.)

Discovery Two: When you enter definitions from the terminal (as you have been doing), your source text† is not saved.

Only the compiled form of your definition is saved in the dic-

†For Beginners

The "source text" is the original version of the definition, such as:

```
    : FOUR-MORE   4 + ;
```

which the compiler translates into a dictionary entry.

tionary. So, what if you want to make a minor change to a word you've already defined? This is where the EDITOR comes in. With the EDITOR, you can save your source text and modify it if you want to.

The EDITOR stores your source text on disk. So before we can really discuss the EDITOR, we'd better introduce the disk and the way the FORTH system uses it.

How FORTH Uses the Disk

Nearly all FORTH systems use disk memory. Even though disk memory is not absolutely necessary for a FORTH system, it's difficult to imagine FORTH without it.

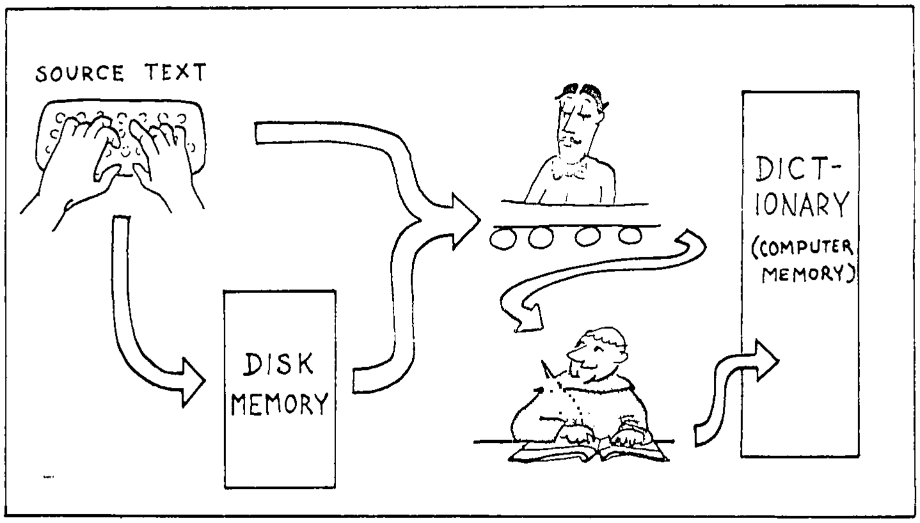
To understand what disk memory does, compare it with computer memory (RAM). The difference is analogous to the difference between a filing cabinet and a rolling card-index.

So far you've been using computer memory, which is like the card index. The computer can access this memory almost instantaneously, so programs that are stored in RAM can run very fast. Unfortunately, this kind of memory is limited and relatively expensive.

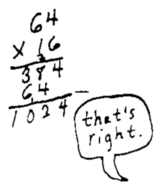
On the other hand, the disk is called a "bulk memory" device because, like a filing cabinet, it can store a lot of information at a much cheaper price per unit of information than the memory inside the computer.

Both kinds of memory can be written to and read from.

The compiler compiles all dictionary entries into computer memory so that the definitions will be quickly accessible. The perfect place to store source text, however, is on the disk, which is what FORTH does. You can either send source text directly from the keyboard to the interpreter (as you have been doing), or you can save your source text on the disk and then later read it off the disk and send it to the text interpreter.



Disk memory is divided into units called "blocks." Many professional FORTH development systems have 500 blocks available (250 from each disk drive). Each block holds 1,024 characters of source text. The 1,024 characters are divided for display into 16 lines of 64 characters each, to fit conveniently on your terminal screen.



```
    180 LIST

          0 ( LARGE LETTER-F)
          1 : STAR   42 EMIT ;
          2 : STARS   0 DO STAR LOOP ;
          3 : MARGIN   CR 30 SPACES ;
          4 : BLIP   MARGIN STAR ;
          5 : BAR   MARGIN 5 STARS ;
          6 : F   BAR BLIP BAR BLIP BLIP  CR ;
          7
          8
          9 F
         10
         11
         12
         13
         14
         15
```

This is what a block looks like when it's listed on your terminal. To list a block for yourself, simply type the block-number and the word LIST, as in:

```
    180 LIST
```

To give you a better idea of how the disk is used, we'll assume that your block 180 contains the sample definitions shown above. Except for line 0, everything should look familiar: these are the definitions you used to print a large letter "F" at your terminal.

Now if you were to type:

```
    180 LOAD
```

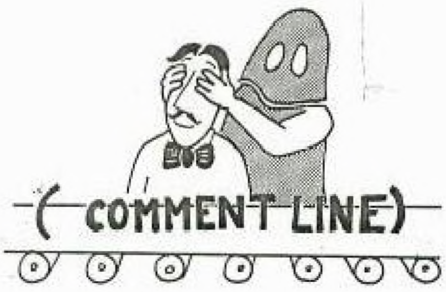
you would send block 180 to the input stream and then on to the text interpreter. The text interpreter does not care where his text comes from. Recognizing the colons, he will have all the definitions compiled.

Notice that we've put our new word F on line 9. We've done this to show that when you load a block, you execute its contents. Simply by typing:

```
    180 LOAD
```

all the definitions will be compiled and a letter "F" will be printed at your terminal.

Now for the unfinished business: line 0. The words inside the parentheses are for humans only; they are neither compiled nor executed. The word ((left parenthesis) tells the text interpreter to skip all the following text up to the terminating right parenthesis. Because (is a word, it must be set off with a space.†



It's good programming practice to identify your application blocks with comments, so that fellow programmers will understand them.

†For Beginners

The closing parenthesis is not a word, it is simply a character that is looked for by (, called a delimiter. (Recall that the delimiter for ." is the closing quote mark.)

Here are a few additional ways to make your blocks easy to read:

1. Separate the name from the contents of a definition by three spaces.

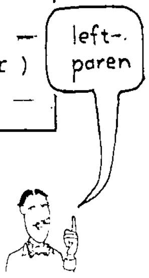
2. Break definitions up into phrases, separated by double spaces.

3. If the definition takes more than one line, indent all but the first line.

4. Don't put more than one definition on a single line unless the definitions are very short and logically related.

To summarize, the three commands we've learned so far that concern disk blocks are:

LIST	(n --)	Lists a disk block.
LOAD	(n --)	Loads a disk block (compiles or executes).
(xxx)	(--)	Causes the string xxx to be ignored by the text interpreter. The character) is the delimiter.



Dear EDITOR[†]

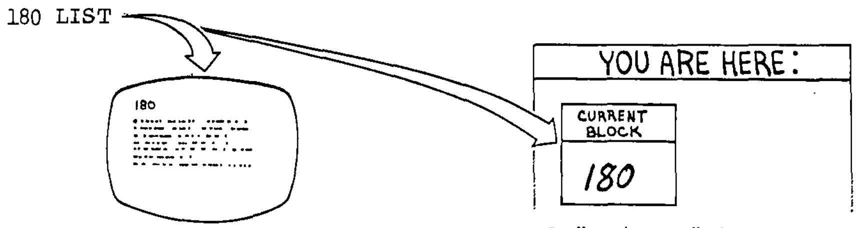
Now you're ready to learn how to put your text on the disk.

First find an empty block[‡] and list it, using the form:

```
    180 LIST
```

When you list an empty block, you'll see sixteen line numbers (0 – 15) running down the side of the screen, but nothing on any of the lines. The "ok" on the last line is the signal that the text interpreter has obeyed your command to list the block.

By listing a block, you also select that block as the one you're going to work on.



The terminal

A "pointer" in computer memory (RAM).

Now that you've made a block "current," you can list it by simply typing the word

```
    L
```

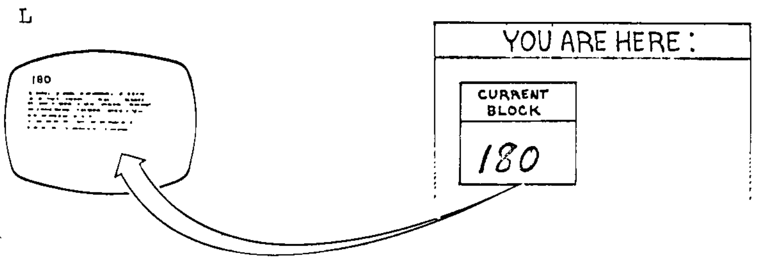
Unlike $\boxed{\text{LIST}}$, $\boxed{\text{L}}$ does not want to be preceded by a block number; instead, it lists the current block.

[†]For Those Whose EDITOR Doesn't Follow These Rules

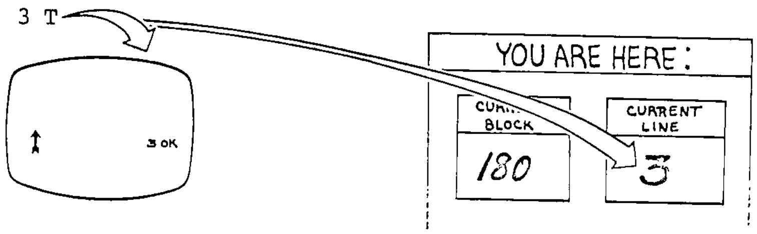
The FORTH-79 Standard does not specify editor commands. Your system may use a different editor; if so, check your system documentation.

[‡]For People at Terminals

If you're using someone else's system, ask them which blocks are available. If you're using your own system, try 180. It should be free (empty).



Now that you have a current block, it's time to select a current line by using the word [T]. Suppose we want to write something on line 3. Type:



[T] lets you select the current line.† It also performs a carriage return, then types the given line (which so far contains nothing). At the end of the line, it reminds you which line you're on:

```
3 T
^
_____________________________________________________________3 ok
```

(Remember, we're underlining the computer's output for the sake of clarity.) The caret at the beginning of the line is the EDITOR's cursor, which points to your current character position. On your terminal the caret might look like this: ↑

---

†For the Curious

Actually, the cursor position, not the line number, serves as the pointer. More on this in a future footnote.

Now that your sights are fixed, you can put some text in the current line by using $\boxed{P}$.

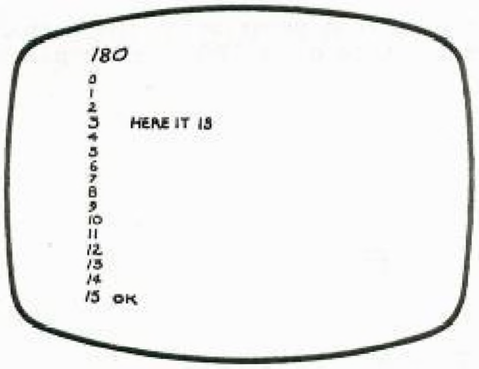
```
    P HERE IT IS RETURN ok
```

$\boxed{P}$ puts the string that follows it (up to the carriage return) on the current line. It does not type out the line. If you don't believe the string is really there, you can type:

```
    3 T
```

or simply:

```
    L
```



Remember that your current position remains the same, so if you were to now type

```
    P THERE IT WENT RETURN ok
```

followed by $\boxed{L}$, you'd see that the latter string had replaced the former on line 3.

Similarly, entering $\boxed{P}$ followed by at least two blank spaces (one to separate the $\boxed{P}$ from the string, the other as the string itself) causes the former string to be replaced by a blank space; in other words, it blanks the line.

In this chapter the symbol "b̸" means that you type a blank space. So to blank a line, type:

```
    Pb̸b̸ RETURN
```

Character Editing Commands

In this section, we'll show you how to insert and delete text within a line.

F

Before you can insert or delete text, you must be able to position the EDITOR's cursor to the point of insertion or deletion. Suppose line 3 now contains

```
    IF MUSIC BE THE FOD OF LOVE
```

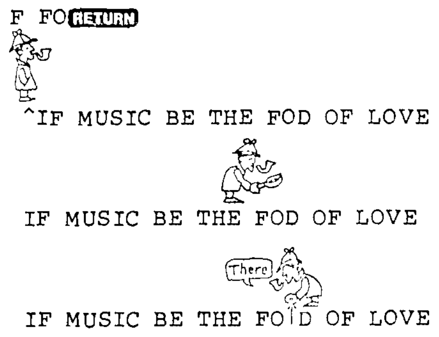
and you want to insert the second "O" in "FOOD," you must first position the cursor after the "FO" like this:

```
    IF MUSIC BE THE FO^D OF LOVE
```

To position the cursor, use the command F, followed by a string, as in

```
    F FO RETURN
```

F searches forward from the current position of the cursor until it finds the given string (in this case "FO"), then places the cursor right after it.



```
   F FO RETURN

    ^IF MUSIC BE THE FOD OF LOVE

    IF MUSIC BE THE FOD OF LOVE

    IF MUSIC BE THE FO|D OF LOVE
```

If you don't know the starting position of the cursor, first type "3 T" to reset the cursor to the start of the line. F then types the line, showing where the cursor is:

```
    IF MUSIC BE THE FO^D OF LOVE                   3 ok
```

$\boxed{I}$

Now that the cursor is positioned where you want it, simply enter:

```
    I ORETURN
```

and $\boxed{I}$ will <u>insert</u> the character "O" just behind the cursor.

```
    IF MUSIC BE THE FOO^D OF LOVE
```

$\boxed{I}$ then types the corrected line, including the cursor:

```
    IF MUSIC BE THE FOO^D  OF LOVE                        3 ok
```

$\boxed{E}$

To <u>erase</u> a string (using the command $\boxed{E}$), you must first <u>find</u> the string, using $\boxed{F}$. For example, if you want to erase the word "MUSIC," first reset the cursor with:

```
    3 TRETURN
```

then type:

```
   F MUSICRETURN
    IF MUSIC^ BE THE FOOD OF LOVE                         3 ok
```

and then simply:

```
    ERETURN
```

$\boxed{E}$ erases the string you just found with $\boxed{F}$.

```
    IF MUSIC BE THE FOOD OF LOVE
```

$\boxed{E}$ then types the line, including the cursor:

```
    IF ^ BE THE FOOD OF LOVE                              3 ok
```

The cursor is now in a position where you can insert another word:

```
 IF ROCK^ BE THE FOOD OF LOVE                        3 ok
```

$\boxed{D}$

The command $\boxed{D}$ finds and deletes a string. It is a combination of $\boxed{F}$ and $\boxed{E}$, giving you two commands for the price of one. For example, if your cursor is here:

```
 IF ROCK^ BE THE FOOD OF LOVE
```

then you can delete "FOOD" by simply typing:

```
D FOOD RETURN
 IF ROCK BE THE ^ OF LOVE                            3 ok
```

Once again, you can insert text at the new cursor position:

```
I CHEESEBURGERS RETURN
 IF ROCK BE THE CHEESEBURGERS^ OF LOVE               3 ok
```

Using $\boxed{D}$ is a little more dangerous than using $\boxed{F}$ and then $\boxed{E}$. With the two-step method, you know exactly what you're going to erase before you erase it.

$\boxed{R}$

The command $\boxed{R}$ replaces a string that you've already found. It is a combination of $\boxed{E}$ and $\boxed{I}$. For instance:

```
F NEED A RETURN
 COMPUTERS NEED A^ TERMINAL                           2 ok
R CAN BE RETURN
 COMPUTERS CAN BE^ TERMINAL                           2 ok
```

$\boxed{R}$ is great when you want to make an insertion in front of a certain string. For example, if your line 0 is missing an "E":

```
 ( SAMPLE DEFINITIONS)              MPTY            0 ok
```

then it's not easy to $\boxed{F}$ your way through all those spaces to get the cursor over to the space before MPTY. Better you should use the following method:

```
 F MPTY RETURN
```

then

```
 R EMPTY RETURN
```

TILL

TILL is the most powerful command for deletion. It deletes everything from the current cursor position up till and including the given string. For example, if you have the line:

BREVITY IS THE SOUL^, THE ESSENCE, AND THE VERY SPARK OF WIT.

(note the cursor position), then the phrase:

```
    TILL SPARK RETURN
```

or even just

```
    TILL K RETURN
```

(since there's only one "K") will produce

```
    BREVITY IS THE SOUL ^OF WIT                          5 ok
```

Has a nicer ring, doesn't it?

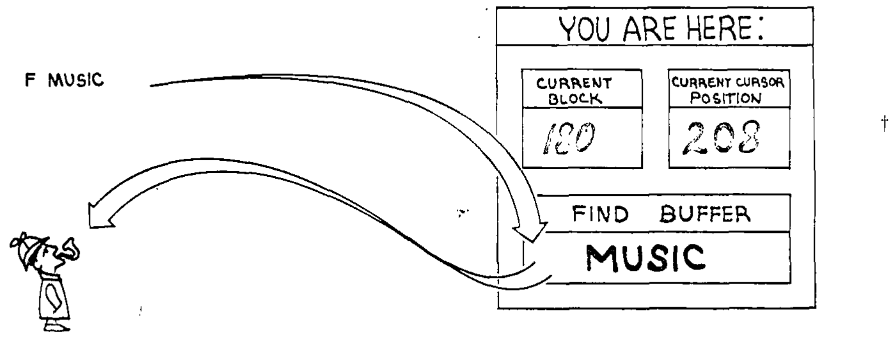
The Find Buffer and the Insert Buffer

In order to use the EDITOR effectively, you really have to understand the workings of its "find buffer" and its "insert buffer."

You may not have known it, but when you typed

```
    F MUSIC RETURN
```

the first thing F did was to move the string "MUSIC" into something called the "find buffer." A buffer, in computer parlance, is a temporary storage place for data. The find buffer is located in computer memory (RAM).



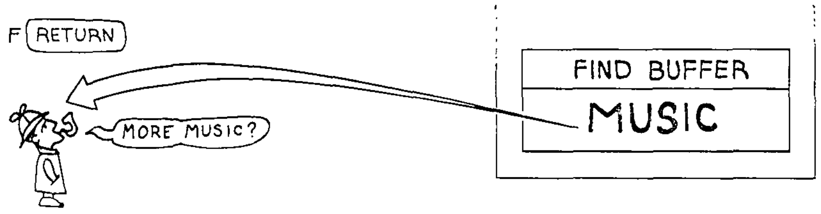
Then $\boxed{F}$ proceeded to search the line for the contents of the find buffer.

Now you will be able to understand the following variation on $\boxed{F}$:

F RETURN

that is, $\boxed{F}$ followed immediately by a return.

This variation causes $\boxed{F}$ to search for the string that is already in the find buffer, left over from the last time you used $\boxed{F}$.



†For the Curious

By keeping the current cursor position, the editor doesn't need to keep a separate pointer for the current line. It simply uses the word $\boxed{/MOD}$. Since there are 64 characters per line, the phrase

```
208 64 /MOD . . 3 16 ok
```

shows the cursor is located at the 16th character in line 3.

What good is this? It lets you find numerous occurrences of the same string without retyping the string. For example, suppose line 8 contains the profundity:

```
    ^THE WISDOM OF THE FUTURE IS THE HOPE OF THE AGES
```

with the cursor at the beginning, and you want to erase the "THE" near the end. Start by typing

```
   F THE␢ RETURN
    THE ^WISDOM OF THE FUTURE IS THE HOPE OF THE AGES 8 ok
```

Now that "THE␢" is in the find buffer, you can simply type a series of single $\boxed{F}$s:

```
   F RETURN
    THE WISDOM OF THE ^FUTURE IS THE HOPE OF THE AGES 8 ok
   F RETURN
    THE WISDOM OF THE FUTURE IS THE ^HOPE OF THE AGES 8 ok
```

etc., until you find the "THE" you want, at which time you can erase it with $\boxed{E}$.†

By the way, if you were to try entering $\boxed{F}$ one more time, you'd get:

```
    F THE NONE
```

This time $\boxed{F}$ cannot find a match for the find buffer, so it returns the word "THE" to you, with the error message "NONE."

Remember we said that $\boxed{D}$ is a combination of $\boxed{F}$ and $\boxed{E}$? Well, that means that $\boxed{D}$ also uses the find buffer.

With the cursor positioned at the beginning of the line and with "THE␢" in the find buffer, you can delete all the "THE"s with single $\boxed{D}$s:

```
   D RETURN
    ^WISDOM OF THE FUTURE IS THE HOPE OF THE AGES      8 ok
   D RETURN
    WISDOM OF ^FUTURE IS THE HOPE OF THE AGES          8 ok
   D RETURN
    WISDOM OF FUTURE IS ^HOPE OF THE AGES              8 ok
   D RETURN
    WISDOM OF FUTURE IS HOPE OF ^AGES                  8 ok
```

†For the Curious

$\boxed{E}$ counts the number of characters in the find buffer and deletes that many characters preceding the cursor.

The other buffer is called the "insert buffer." It is used by $\boxed{\text{I}}$. Simply typing:

```
   IRETURN
```

will insert the contents of the insert buffer at the current cursor position. The following experiment will demonstrate how you might use both buffers at the same time. Suppose line 14 contains...

```
   ^THE YONDER, THE DANUBE, AND THE MAX             14 ok
```

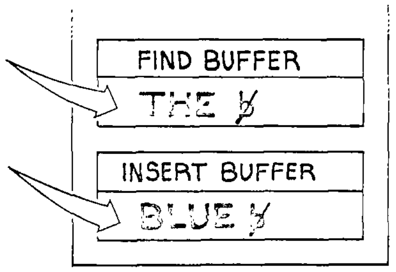
Now position the cursor:

```
  F THEb̸RETURN
   THE ^YONDER, THE DANUBE, AND THE MAX             14 ok
```

and insert:

```
  I BLUEb̸RETURN
   THE BLUE ^YONDER, THE DANUBE, AND THE MAX        14 ok
```

You have now loaded both buffers like so:



Now type:

```
  FRETURN
   THE BLUE YONDER, THE ^DANUBE, THE MAX            14 ok
```

and:

```
  IRETURN
   THE BLUE YONDER, THE BLUE ^DANBUE, THE MAX       14 ok
```

and again:

```
  FRETURN
   THE BLUE YONDER, THE BLUE DANUBE, THE ^MAX       14 ok
  IRETURN
   THE BLUE YONDER, THE BLUE DANUBE, THE BLUE ^MAX  14 ok
```

This is what a computer scientist would call "spiffy."

Line Editing Commands

Now that we've shown you how to move letters and words around, we'll show you how to move whole lines around.

$\boxed{P}$

The word $\boxed{P}$, which we introduced before, uses the very same insert buffer that $\boxed{I}$ uses. Assuming that you still have "BLUE" in your insert buffer from the previous example and that line 14 is still your current line, then typing:

    P RETURN

will replace the old line 14 with the contents of the insert buffer, so that line 14 now contains only the single word:

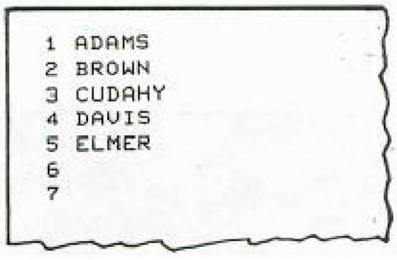
    BLUE

To quickly review, you have now learned three ways to use $\boxed{P}$:

1)	P ALL THIS TEXT RETURN	puts the string in the insert buffer, then in the current line.
2)	P␢␢ RETURN	blanks the insert buffer, then blanks the current line.
3)	P RETURN	puts the contents of the insert buffer in the current line.

$\boxed{U}$

A very similar word is $\boxed{U}$. It places the contents of the insert buffer under the current line. For example, suppose your block contains:



```
1 ADAMS
2 BROWN
3 CUDAHY
4 DAVIS
5 ELMER
6
7
```

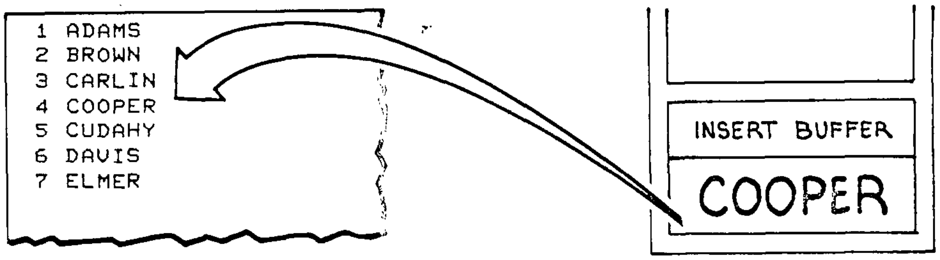
If you move your cursor to line 2 with:

```
2 T
^BROWN                                        3 ok
```

and then type:

```
U CARLIN RETURN ok
U COOPER RETURN ok
```

you'll get:



Instead of replacing the current line, [U] squeezes the contents of the insert buffer in below the current line, pushing all the lines below it down. If there were anything in line 15, it would roll off and disappear.

It's easier to use [U] than [P] when you're adding successive lines. For example:

```
1 T P ADAMS RETURN ok
U BROWN RETURN ok
U CUDAHY RETURN ok
U DAVIS RETURN ok
etc.
```

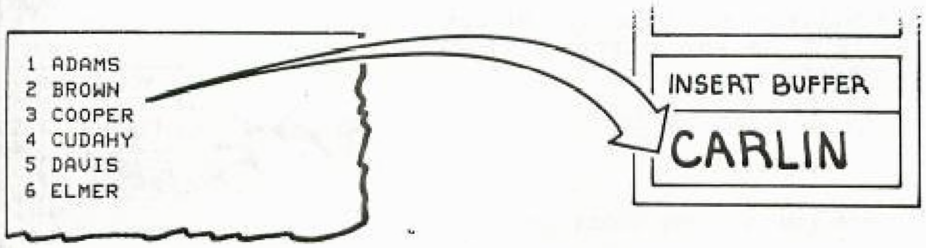
The three ways of using [P] also apply to [U].

[X]

[X] is the opposite of [U]; it extracts the current line. Using the above example, if you make line 3 current (with the phrase "3 T"), then by entering:

```
X RETURN
```

you extract line 3 and move the lower lines up.



As you see, $\boxed{X}$ also moves the extracted line into the insert buffer. This makes it easy to move the extracted line anywhere you want it. For example, the combination:

```
    9 T RETURN
```

and:

```
    P RETURN
```

would now put "CARLIN" on line 9.

<u>Miscellaneous EDITOR Comands</u>

$\boxed{\text{WIPE}}$

The word $\boxed{\text{WIPE}}$ blanks an entire block. You can use $\boxed{\text{WIPE}}$ to ensure that there will not be any strange characters which might keep a block from being loaded.

If your system doesn't have $\boxed{\text{WIPE}}$, another way to blank an entire block is this: first enter

```
    0 T RETURN
```

then hit

```
    X
```

sixteen times.

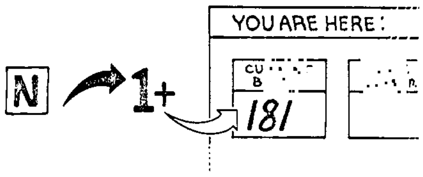
N and B

When you type the word N, you add one to the current block number.

Thus the combination:

```
    N L
```

causes the <u>next</u> block to be listed.

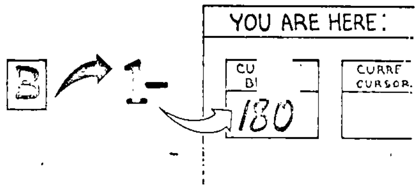


Similarly, the word B subtracts one from the current block number.

Thus the combination:

```
    B L
```

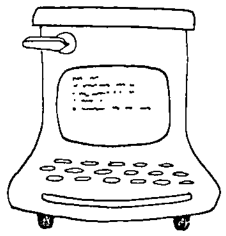
lets you list one block <u>back</u>.



FLUSH

We can't say too much about this word until we discuss how the FORTH "operating system" converses with the disk, but for now you should know this: FLUSH[†] assures you that any change you've made to a block really gets written to the disk.

Say you've made some changes to a block, then you turn off the computer. When you come back tomorrow and list the block, it may seem as though you never made the changes at all. The operating system simply didn't get around to writing the corrected block to the disk before you turned off the computer. The same thing could happen if you were to load your application and then crash the system before it could write the changes to disk.



[†]FORTH-79 Standard

In the Standard, the name for this word is SAVE-BUFFERS.

So always enter $\boxed{\text{FLUSH}}$ before removing the disk, cycling power, or trying something dangerous. Some programmers habitually $\boxed{\text{FLUSH}}$ after every change without even thinking about it.

$\boxed{\text{COPY}}$

The word $\boxed{\text{COPY}}$ lets you <u>copy</u> one block to another, displacing whatever was in the destination block. You use it in this form:

```
    from to COPY
```

For example, entering:

```
    153 200 COPY
```

will copy whatever is in block 153 into block 200.

Make it a habit to $\boxed{\text{FLUSH}}$ after every $\boxed{\text{COPY}}$.

$\boxed{\text{S}}$

$\boxed{\text{S}}$ is an expanded version of $\boxed{\text{F}}$. It lets you <u>search</u> for a given string in and beyond your current block into the following blocks, up to the block that you specify.

For example, if your current block is 180, and you type:

```
    185 S TREASURE
```

then $\boxed{\text{S}}$ will search for "TREASURE" in blocks 180 thru 184. If it finds "TREASURE" in, say, block 183, it will type:

```
    THIS MOMENT THAT WE TREASURE^ TOGETHER        7 183 ok
```

giving both the block and the line number.

The block number with which you precede the word $\boxed{\text{S}}$ represents the next block <u>after</u> the last one you want searched. There is a reason for this, but it won't make sense until a later chapter.

$\boxed{M}$

$\boxed{M}$ lets you move an individual line (or group of lines) from one block to another. To move a line to another block, first make the line current with

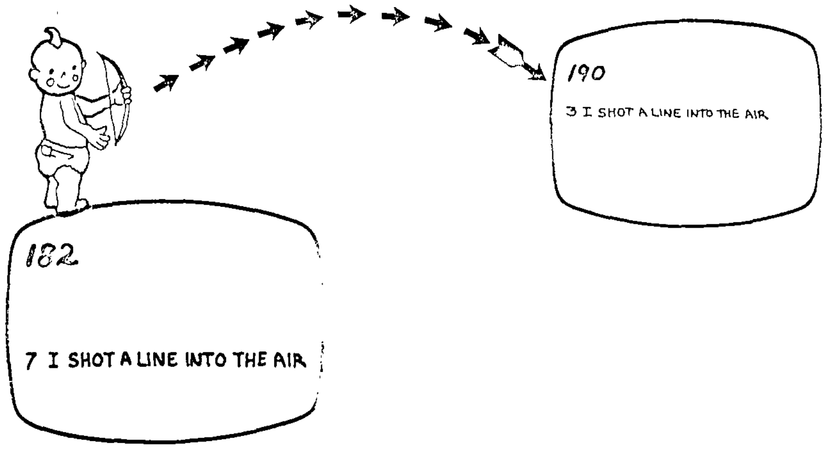
```
    182 LIST
```

then

```
   7 T
   ^I SHOT A LINE INTO THE AIR                            7 ok
```

Then enter the destination block and the number of the line under which you want the line inserted, followed by the word $\boxed{M}$:

```
190 2 M
```



The line of text in the current block (block 182) moves down to the next line. So to move three consecutive lines, simply enter

```
    190 2 M
    190 3 M
    190 4 M
```

$\boxed{\wedge}$

You can type the caret character instead of RETURN to indicate the end of a character string, so that you can get more than one command on a line.

For example, you could type:

```
    D FRUIT^ I NUTSRETURN
```

all on the same line, and get the same result as if you had typed:

```
    D FRUITRETURN
```

and:

```
    I NUTSRETURN
```

That's it for the EDITOR commands. Because FORTH is naturally flexible, and because users can define their own EDITOR commands if they want to, the set of EDITOR commands in your system may vary from the set presented here. This chapter closes with a review of all the commands we've talked about.

One final observation about the EDITOR: it is not a program, as it might be in another language. It is rather a collection of words. The EDITOR, in fact, is called a "vocabulary." We'll discuss the significance of vocabularies in a later chapter.

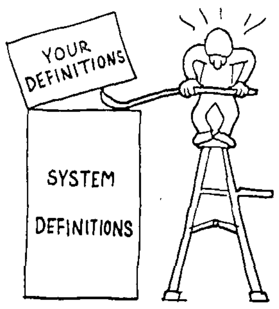
Getting $\boxed{\text{LOAD}}$ed

Now that you've learned to edit your definitions into a block, it's time to load them. But consider for a moment: each time you load definitions, you increase the size of your dictionary.

For example, let's say you write a definition for something you call 1FUNCTION, edit it into an available block, and load it. You test it and realize you forgot a $\boxed{\text{SWAP}}$. So you fix the source text with the EDITOR commands, then load the block again. It works!

Now in the same block you edit in a definition of something you call 2FUNCTION and load the block again. This time, you get it right on the first try. But what does your dictionary look like? From loading this block three times, you've got three versions of 1FUNCTION in there. The simplest way to avoid this problem is to use the word

EMPTY

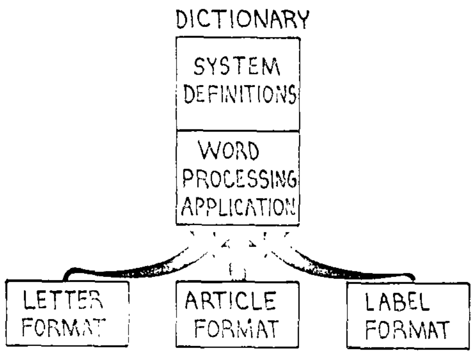


EMPTY "forgets" all the definitions that you yourself have defined (not system definitions).† If you put EMPTY at the beginning of the block, you will start with a clean slate each time you load.

For example:

```
0 ( SOLUTIONS -- QUIZZIE 2-B)      EMPTY
1 : 2B1    * + ;
2 : 2B2    4 * -  6 / + ;
3
```

Sometimes you don't want to get rid of your whole application, only part of it. Suppose you were to write a word processing application (so you can enter text, edit it in memory, then output it to a printer). After you've finished the basic application, you want to add variations, so it can use one format for correspondence, another format for magazine articles, and another format for address labels.



†For People on a Multiprogrammed System

EMPTY "forgets" your own personal extension of the dictionary, not anyone else's.

In FORTH these three variations are called "overlays" because they are mutually exclusive and can be made to replace each other. Here's how.

The basic word processing application should begin with EMPTY. The last definition should be a name only, such as

```
    : VARIATIONS ;
```

This is called a "null definition" because it does nothing but mark a place in your portion of the dictionary.

Then at the beginning of each variation block, include the expression

```
    FORGET VARIATIONS              : VARIATIONS ;
```

Now when you load one variation, it FORGETs back to the null definition, compiles a new null definition, and then compiles the variation's definitions. When you load the other variation, you replace the first overlay with the second overlay.

One more trick: what if the source text for your application takes more than one block? The best solution is to let one block load the other blocks. For example, your "load block" might contain:

```
0 ( MY APPLICATION)
1
2 180 LOAD   181 LOAD   182 LOAD
```

It's much better to let a single load block LOAD all the related blocks than to let each block load the next one in a chain.

Now you know the ropes of disk storage. You'll probably want to edit most of the remaining examples and problems in this book into disk blocks rather than straight from the keyboard to the interpreter, especially the longer ones. It's just easier that way.

A Handy Hint — When a Block Won't LOAD

On some FORTH systems, the following scenario may sometimes happen to you: you load some new definitions from a block, but when you try to execute them, FORTH doesn't seem to have ever heard of them (responding with a "?").

First you want to check whether any or all of your definitions were actually compiled into the dictionary. To do this, enter an apostrophe followed by a space, then the name of the word, then a ., as in

```
' THINGAMAJIG .RETURN
```

If . prints a number, then the definition is compiled, but if FORTH responds

```
THINGAMAJIG ?
```

then it isn't. There are two possible reasons for part of a block not getting compiled:

1) You made a typing error that keeps FORTH from being able to recognize a word. For instance, you may have typed

```
(COMMENT LINE)
```

without a space after (. This type of error is easy to find and correct because FORTH prints the name of any word it doesn't understand, like this:

```
180 LOADRETURN (COMMENT ?
```

2) There is a non-printing character (one you can't see)† somewhere in the block. To find a non-printing character, enter this:

```
0 TRETURN
1 TRETURN
2 TRETURN     etc.
```

If a line contains any non-printing characters, the "ok" at the end of the line will not line up with the "ok"s at the ends of the other lines, because non-printing characters don't print spaces. For any such line, reenter the entire line (using P).

†For Experts

The "null" character (ASCII 0) is the culprit. On most FORTH systems, null is actually a defined word, synonymous with EXIT, a word we will discuss in Chap. 9.

A Handy Hint

A Better Non-destructive Stack Print

Now that you know how to load longer definitions from a disk block, here's an improved version of .S which displays the contents of the stack non-destructively without displaying the "stack-empty" number.

This version uses an additional word called DEPTH, which returns the number of values on the stack. (Follow it with $\boxed{.}$.)†

If you're a beginner, you might want to enter these two definitions into a special block all by themselves so you can load them any time you want them.

```
( NON-DESTRUCTIVE STACK PRINT)

: DEPTH   S0 @  'S - 2/ 2-  ;
: .S   CR  DEPTH  IF
     'S S0 @ 4 -   DO I @ . -2 +LOOP
          ELSE ." Empty " THEN ;


```

†FORTH-79 Standard

The Standard word set includes $\boxed{\text{DEPTH}}$.

Here's a list of the FORTH words we've covered in this chapter:

LIST	(n --)	Lists a disk block.
LOAD	(n --)	Loads a disk block (compiles or executes).
(xxx)	(--)	Causes the string xxx to be ignored by the text interpreter. The character) is the delimiter.
FLUSH	(--)	Forces any modifications that have been made to a block to be written to disk.
COPY	(source dest --)	Copies the contents of the source block to the destination block.
WIPE	(--)	Sets the contents of the current block to blanks.
FORGET xxx	(--)	Forgets all definitions back to and including xxx.
EMPTY	(--)	Forgets the entire contents of the user's dictionary.

Editing Commands -- Line Operators

T	(n --)	Types the line.
P Pƀƀ or P xxx	(--)	Copies the given string, if any, into the insert buffer, then puts a copy of the insert buffer in the current line.
U Uƀƀ or U xxx	(--)	Copies the given string, if any, into the insert buffer, then puts a copy of the insert buffer in the line under the current line.
M	(block line --)	Copies the current line into the insert buffer, and moves a copy of the insert buffer into the line under the specified line in the destination block.

X	(--)	Copies the current line into the insert buffer and ex-tracts the line from the block.

Editing Commands -- String Operators

F or F xxx	(--)	Copies the given string, if any, into the find buffer, then finds the string in the current block.
S or S xxx	(n --)	Copies the given string, if any, into the find buffer, then searches the range of blocks, starting from the current block and ending with n-1, for the string.
E	(--)	To be used after F. Erases as many characters as are currently in the find buffer, going backwards from the cursor.
D or D xxx	(--)	Copies the given string, if any, into the find buffer, finds the next occurrence of the string within the current line, and deletes it.
TILL or TILL xxx	(--)	Copies the given string, if any, into the find buffer, then deletes all characters starting from the current cursor position up till and including the string.
I or I xxx	(--)	Copies the given string, if any, into the insert buffer, then inserts the contents of the insert buffer at the point just behind the cursor.
R or R xxx	(--)	Combines the commands E and I to replace a found string with a given string or the contents of the insert buffer.
↑ or ^	(--)	Indicates the end of the string to be placed in a buffer.

Review of Terms

Block	in FORTH, a division of disk memory containing up to 1024 characters of source text.
Buffer	a temporary storage area for data.
~~Disk~~	a disk that has been coated with a magnetic material so that, as in a tape recorder, a "head" can write or read data on its surface as the disk spins.
EDITOR	a vocabulary which allows a user to enter and modify text on the disk.
Find buffer	in FORTH's EDITOR, a memory location in which the string that is to be searched for is stored. Used by $\boxed{\text{F}}$, $\boxed{\text{E}}$, $\boxed{\text{D}}$, $\boxed{\text{TILL}}$, and $\boxed{\text{S}}$.
Insert Buffer	in FORTH's EDITOR, a memory location in which the string that is to be inserted is stored. Used by $\boxed{\text{I}}$, $\boxed{\text{P}}$, and $\boxed{\text{U}}$. In addition, $\boxed{\text{X}}$ moves the line that it deletes into the insert buffer.
Load block	one block which, when loaded, itself loads the rest of the blocks for an application.
Null Definition	a definition that does nothing, written in the form: `: NAME ;` that is, a name only will be compiled into the dictionary. A null definition serves as a "bookmark" in the dictionary, for $\boxed{\text{FORGET}}$ to find.
Overlay	a portion of an application which, when loaded, replaces another portion in the dictionary.
Pointer	a location in memory where a number can be stored (or changed) as a reference to something else.
Source text	in FORTH, the written-out form of a definition or definitions in English-like words and punctuation, as opposed to the compiled form that is entered into the dictionary.

Problems — Chapter 3

1. a) Enter your definitions of GIFT, GIVER and THANKS from Probs. 1 and 3 of Chap. 1 into a block, then load and execute THANKS.

   b) Using the EDITOR, change the person's name in the definition of GIVER, then load and execute THANKS again. What happens this time?

2. Try loading some of your mathematical definitions from Chap. 2 into an available block, then load it. Fool around.