SpeedScript

The Word Processor for Atari Computers

Charles Brannon

A COMPUTE! Books Publication

$10.95
SPEEDSCRIPT
The Word Processor for Atari Computers

Charles Brannon
“SpeedScript 3.0: All Machine Language Word Processor for Atari” was originally published in COMPUTE! magazine, May 1985, copyright 1985, COMPUTE! Publications, Inc.

Copyright 1985, COMPUTE! Publications, Inc. All rights reserved.

Reproduction or translation of any part of this work beyond that permitted by Sections 107 and 108 of the United States Copyright Act without the permission of the copyright owner is unlawful.

Printed in the United States of America

10 9 8 7 6 5 4 3 2

ISBN 0-87455-003-3

The author and publisher have made every effort in the preparation of this book to insure the accuracy of the programs and information. However, the information and programs in this book are sold without warranty, either express or implied. Neither the author nor COMPUTE! Publications, Inc. will be liable for any damages caused or alleged to be caused directly, indirectly, incidentally, or consequentially by the programs or information in this book.

The opinions expressed in this book are solely those of the author and are not necessarily those of COMPUTE! publications, Inc.

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is part of ABC Consumer Magazines, Inc., one of the ABC Publishing Companies, and is not associated with any manufacturer of personal computers. Atari 400, 800, 600XL, 800XL, 1200XL, and XE are trademarks of Atari, Inc.
Contents

Foreword ................................................................. v

Chapter 1. Using SpeedScript ................................. 1
  SpeedScript 3.0: All Machine Language
  Word Processor for the Atari ............................ 3

Chapter 2. Entering SpeedScript ......................... 25
  The Machine Language Editor: MLX ..................... 27
  The Automatic Proofreader ............................... 34
  SpeedScript Program Listings ........................... 37

Chapter 3. SpeedScript Source Code .................... 67
  Atari Source Code ........................................ 69

Index ................................................................. 113
Order Coupon for Disk ........................................ 115
Foreword

*SpeedScript* is the most popular program ever published by COMPUTE! Publications. Ever since it first appeared in the January 1984 issue of COMPUTE!'s Gazette, the letters have been pouring in. People wanted to know more about the program and word processing, and they had countless suggestions about how to make *SpeedScript* better.

The result is *SpeedScript 3.0*, an even more powerful word processor for all eight-bit Ataris (including the 400/800, 600XL/800XL, 1200XL, and new XE series). Enhanced with additional commands and features, this all machine language word processor gives you all the things you expect from a commercial software package. You can write, edit, format, and print anything from memos to novels on your Atari. With a few keystrokes you can change the color of the screen and its text to whatever combination best suits you.

It's easy to add or delete words, letters, even whole paragraphs. You can search through an entire document and find every occurrence of a particular word or phrase, then replace it with something new. Of course, when you finish writing, you can save your work to tape or disk.

The ability to quickly change the appearance of a printed document is one of the things that make word processing so efficient. *SpeedScript* lets you alter the margins, page length, spacing, page numbers, page width, as well as set up headers and footers at the top and bottom of the paper.

And once you've formatted your document, you'll find enough print features to make even the most demanding writer happy. With *SpeedScript*, you can start printing at any page, force the printer to create a new page at any time, even make it wait while you put in another sheet of paper. Underlining and centering are simple. If you want to get fancy, you can use your printer's codes to create graphics symbols or logos. And if you're writing something really long—perhaps a novel or term paper—*SpeedScript* lets you link any number of files so that they print out as one continuous document.

In addition to the *SpeedScript* program for the Atari, you'll find complete documentation and a keyboard map in this book. *SpeedScript*'s source code has also been included for
your examination. By studying it, you'll see exactly how the program is put together.

"The Machine Language Editor: MLX" makes typing in the program easier. MLX almost guarantees that you'll have an error-free copy of the program the first time you type it in. If you prefer to purchase a copy of SpeedScript on disk rather than type it in, just use the convenient coupon in the back, or call toll-free 1-800-334-0868.

SpeedScript is an exceptionally easy-to-use and powerful word processor that will meet all your writing needs.
Chapter 1
Using SpeedScript
SpeedScript 3.0
All Machine Language Word Processor for the Atari

SpeedScript has become one of the most popular word processors for the Commodore 64, VIC-20, and Apple computers. And now SpeedScript has been translated to run on all eight-bit Ataris with at least 24K, with either disk or cassette (including the 400, 800, 600XL with memory expansion, 800XL, 1200XL, and new XE series). SpeedScript compares favorably with commercial word processors and has some features never seen before in an Atari word processor. It represents unique value in a type-in program.

SpeedScript 3.0, though compact in size (8K), has many features found on commercial word processors. SpeedScript is also very easy to learn and use. You type in everything first; preview and make corrections on the screen; insert and delete words, sentences, and paragraphs; then print out an error-free draft, letting SpeedScript take care of things like margins, centering, headers, and footers.

Typing In SpeedScript
Atari SpeedScript is the longest machine language program we’ve ever published, but COMPUTE!’s "MLX" entry system helps you type it right the first time. MLX can detect most errors people make when entering numbers. (See the instructions for using MLX in chapter 2.) MLX also lets you type SpeedScript in more than one sitting. Although the program listing is lengthy, we guarantee the effort will be worthwhile.

After you run the Atari version of MLX, answer the first two questions like this:
Starting Address? 7936
Ending Address? 16229
Run/Init Address 7936

Next, you’ll be asked "Tape or Disk." SpeedScript can be saved as either a binary file on disk or as a boot tape. Press T for use with a tape drive. If you press D for disk, you’ll be asked "Boot Disk or Binary File." Press F to select the Binary File option. Although you could save SpeedScript as an auto-booting disk, it makes no sense, because such a disk cannot...
SpeedScript

contain DOS, which is necessary for file-oriented disk access.

The screen will then show the first prompt, the number 7936 followed by a colon (:). Type in each three-digit number shown in the listing. You do not need to type the comma shown in the listing. MLX inserts the comma automatically.

The last number you enter in each line is a checksum. It represents the values of the other numbers in the line summed together. If you type the line correctly, the checksum calculated by MLX and displayed on the screen should match the checksum number in the listing. If it doesn’t match, you will have to retype the line. MLX is not foolproof, though. It’s possible to fool the checksum by exchanging the positions of the three-digit numbers. Also, an error in one number can be offset by an error in another. MLX will help catch your errors, but you must still be careful.

**Typing in Multiple Sittings**

If you want to stop typing the listing at some point and pick up later, press CTRL-S and follow the screen prompts. (For disk, MLX will ask you to specify a filename; do not use AUTORUN.SYS until the entire listing is typed in.) Remember to note the line number of the last line you entered. When you are ready to continue typing, load MLX, answer the prompts as you did before, then press CTRL-L. For a boot tape, be sure the cassette is in the tape drive and rewound. For a binary disk file, MLX asks for the filename you gave to the partially typed listing. After the LOAD is complete, press CTRL-N and tell MLX the line number where you stopped. Now continue typing as before.

When you finish all typing, MLX automatically prompts you to save the program. For disks with Atari DOS 2.0 or 3.0, save the completed program with the filename AUTORUN.SYS. This will allow SpeedScript to load and run automatically when the disk is booted.

At this point, MLX has saved either a boot tape or binary disk file. To load your boot tape, remove all cartridges, rewind the tape, and hold down the START button while turning on the power. (On the 600XL, 800XL, and XE series, disable BASIC by holding down both START and OPTION while turning on the power.) When the computer turns on, you’ll hear a single beep tone. (On the XL and XE series, make sure the volume is turned up on your TV or monitor.) Press PLAY
Using *SpeedScript*

on the tape drive, then press any key on the keyboard to start the load. *SpeedScript* will automatically run once the boot is successfully completed.

To use *SpeedScript* with an Atari DOS disk, you must save or copy it on a disk which also contains DOS.SYS and DUP.SYS. Since you've saved *SpeedScript* as AUTORUN.SYS, it will automatically load and run when you turn on your computer with this disk in the drive. (On the 600XL, 800XL, and XE series, disable BASIC by holding down OPTION when switching on the computer.) *SpeedScript* must always be named AUTORUN.SYS in order to load properly with Atari DOS. If you want to prevent it from automatically running for some reason, you can save it with another name, then rename it AUTORUN.SYS later.

If you're using Optimized System Software's OS/A+ DOS or a compatible successor, you can give *SpeedScript* any filename you like. Just use the LOAD command from DOS, and *SpeedScript* will automatically run. Or you can give it a filename with the extension .COM, such as SPEED.COM. Then you can start up by just typing SPEED at the DOS prompt. You can also write a simple batch file to boot up *SpeedScript* automatically. Some enhanced DOS packages like Optimized System Software's DOS XL may use so much memory that they conflict with *SpeedScript*. In this case, you'll need either to use Atari DOS instead on your *SpeedScript* disks or to reassemble the source code at a higher address to avoid conflicts.

Note: The AUTORUN.SYS file on your DOS master disk is responsible for booting up the 850 Interface Module for RS-232 communications. There is no easy way to combine the 850 boot program with *SpeedScript*, so you can't access the R: device. We'll show you later how to transfer files over a modem or print to a serial printer.

If you prefer, Atari *SpeedScript* is available for purchase on disk. To order the disk, use the coupon in the back of this book or call COMPUTE! Publications toll-free at 800-334-0868.

**Entering Text**

When you run *SpeedScript*, the screen colors change to black on white. The first line on the screen is black with white letters. *SpeedScript* presents all messages on this command line. The remaining 18 lines of the screen are used to enter, edit,
and display your document. *SpeedScript* makes use of a special, but little-used, Atari character mode that permits larger, more readable characters with true lowercase descenders. The screen still shows up to 40 columns; only five rows are sacrificed. We think you'll agree that this is the most readable text you've ever seen on an Atari—perfect for word processing. (Technical note: *SpeedScript* starts at $1F00, and the AN- TIC 3 character set is embedded at $2000.)

The cursor, a blinking square, shows where the next character you type will appear on the screen. *SpeedScript* lets you move the cursor anywhere within your document, making it easy to find and correct errors.

To begin using *SpeedScript*, just start typing. When the cursor reaches the right edge of the screen, it automatically jumps to the beginning of the next line, just as in BASIC. But unlike BASIC, *SpeedScript* never splits words at the right edge of the screen. If a word you're typing won't fit at the end of one line, it's instantly moved to the next line. This feature, called *word-wrap*, or *parsing*, also helps to make your text more readable.

**Scrolling and Screen Formatting**

When you finish typing on the last screen line, *SpeedScript* automatically scrolls the text upward to make room for a new line at the bottom. Imagine the screen as an 18-line window on a long, continuous document. If you've unplugged all cartridges or disabled BASIC as described above, there's room in memory for 3328 characters of text with 24K RAM and up to 27,904 characters on a 48K machine. (Unfortunately, *SpeedScript* 3.0 cannot make use of the extra memory available in the XL and XE series.) An additional 2K of text memory is available if *SpeedScript* is loaded from a boot tape. To check at any time how much unused space is left, press **CTRL-U** (hold down the CTRL key while pressing the U key). The number appearing in the command line indicates how much unused room remains for characters of text.

If you're used to a typewriter, you'll have to unlearn some habits if this is your first experience with word processing. Since the screen is only 40 columns wide, and most printers have 80-column carriages, it doesn't make sense to press RETURN at the end of each line as you do on a typewriter. *SpeedScript*'s word-wrap takes care of this automatically. Press
RETURN only when you want to force a carriage return to end a paragraph or limit the length of a line. A return-mark appears on the screen as a crooked left-pointing arrow.

**Using the Keyboard**

Most features are accessed with control key commands—you hold down CTRL while pressing another key. In this book, control key commands are abbreviated CTRL-\textit{x} (where \textit{x} is the key you press in combination with CTRL). An example is the CTRL-U mentioned above to check on unused memory. CTRL-E means hold down CTRL and press E. Sometimes you must also hold down the OPTION button to select a special option of a command, such as OPTION-CTRL-H. Other keys are referenced by name or function, such as DELETE/BACK S for the backspace key, CTRL-CLEAR for the clear-screen key, and \textit{cursor left} or CTRL-\textit{+} for the cursor-left key. (See the "Keyboard Map," page 18, for a summary of the keyboard commands.)

Some keys let you move the cursor to different places in the document to make corrections or scroll text into view. You can move the cursor by character, word, sentence, or paragraph. Here's how to control the cursor:

- The \textit{cursor left/right} keys (CTRL-\textit{+} and CTRL-\textit{*}) work as usual; pressing CTRL-\textit{*} moves the cursor right (forward) one space, and CTRL-\textit{+} moves the cursor left (backward) one space.
- The \textit{cursor up/down} keys (CTRL-minus and CTRL-\textit{=}) move the cursor to the beginning of either the next or previous sentence. Press CTRL-minus to move the cursor up (backward) to the beginning of the previous sentence. Press CTRL-\textit{=} to move the cursor down (forward) to the beginning of the next sentence.
- \textit{SHIFT}-+ moves the cursor left (backward) to the beginning of the previous word. \textit{SHIFT}-* moves the cursor right (forward) to the beginning of the next word. If you get confused, just look at the arrows on the keys for a reminder.
- \textit{SHIFT}-\textit{-} moves the cursor up (backward) to the beginning of the previous paragraph. \textit{SHIFT}-\textit{=} moves the cursor down (forward) to the beginning of the next paragraph. Again, look at the arrows on these keys for a reminder. A paragraph always ends with a return-mark.
• The **START** button, pressed once, moves the cursor to the top (start) of the screen without scrolling. Pressed twice, it moves the cursor to the start of the document.

• **CTRL-Z** moves the cursor to the end of the document, scrolling if necessary. It’s easy to remember since Z is at the end of the alphabet.

For special applications, if you ever need to type the actual character represented by a command or cursor key, press **ESC** before typing the CTRL key. Press ESC twice to get the ESCape character, CHR$(27).

**Correcting Your Typing**

Sometimes you’ll have to insert some characters to make a correction. Use **CTRL-INSERT** to open up a single space, just as in BASIC. Merely position the cursor at the point where you want to insert a space, and press CTRL-INSERT.

It can be tedious to use CTRL-INSERT to open up enough space for a whole sentence or paragraph. For convenience, *SpeedScript* has an insert mode that automatically inserts space for each character you type. In this mode, you can’t type over characters; everything is inserted at the cursor position. To enter insert mode, press **CTRL-I**. To cancel insert mode, press CTRL-I again. To let you know you’re in insert mode, the black command line at the top of the screen turns blue.

Insert mode is the easiest way to insert text, but it can become too slow when inserting near the top of a very long document because it must move *all* the text following the cursor position. So *SpeedScript* has even more ways to insert blocks of text.

One way is to use the **TAB** key. It is programmed in *SpeedScript* to act as a five-space margin indent. To end a paragraph and start another, press RETURN twice and press TAB. TAB always inserts; you don’t need to be in insert mode. You can also use TAB to open up more space than CTRL-INSERT. (You cannot set or clear tab stops in *SpeedScript* as you can with the normal screen editor.) No matter how much space you want to insert, each insertion takes the same amount of time. So the TAB key can insert five spaces five times faster than pressing CTRL-INSERT five times.

There’s an even better way, though. Press **SHIFT-INSERT** to insert 255 spaces (it does not insert a line; use RETURN for
that). You can press it several times to open up as much space as you need. And SHIFT-INSERT is fast. It inserts 255 spaces as fast as CTRL-INSERT opens up one space. Now just type
the text you want to insert over the blank space. (You don’t
want to be in CTRL-I insert mode when you use this trick;
that would defeat its purpose.)

Since the DELETE/BACK S key (backspace) is also slow
when working with large documents (it, too, must move all
text following the cursor), you may prefer to use the cursor-left
key to backspace when using this method.

After you’ve finished inserting, there may be some in-
serted spaces left over that you didn’t use. Just press
SHIFT-DELETE/BACK S. This instantly deletes all extra
spaces between the cursor and the start of following text. It’s
also useful whenever you need to delete a block of spaces for
some reason.

**Erasing Text**

Press DELETE/BACK S by itself to erase the character to the
left of the cursor. All the following text is pulled back to fill
the vacant space.

Press CTRL-DELETE/BACK S to delete the character on
which the cursor is sitting. Again, all the following text is
moved toward the cursor to fill the empty space.

These keys are fine for minor deletions, but it could take
all day to delete a whole paragraph this way. So SpeedScript
has two commands that can delete an entire word, sentence,
or paragraph at a time. CTRL-E erases text after (to the right
of) the cursor position, and CTRL-D deletes text behind (to the
left of) the cursor.

To use the **CTRL-E erase mode**, first place the cursor at
the beginning of the word, sentence, or paragraph you want to
erase. Then press CTRL-E. The command line shows the mes-
 sage “Erase (S,W,P): RETURN to exit.” Press S to erase a sen-
tence, W for a word, or P for a paragraph. Each time you press
one of these letters, the text is quickly erased. You can keep
pressing S, W, or P until you’ve erased all the text you wish.
Then press RETURN to exit the erase mode.

The **CTRL-D delete mode** works similarly, but deletes
only one word, sentence, or paragraph at a time. First, place
the cursor after the word, sentence, or paragraph you want to
delete. Then press CTRL-D. Next, press S, W, or P for sentence, word, or paragraph. The text is immediately deleted and you return to editing. You don’t need to press RETURN to exit the CTRL-D delete mode unless you pressed this key by mistake. *(In general, you can escape from any command in SpeedScript by simply pressing RETURN.*) CTRL-D is most convenient when the cursor is already past what you’ve been typing.

**The Text Buffer**

When you erase or delete with CTRL-E or CTRL-D, the text isn’t lost forever. *SpeedScript* remembers what you’ve removed by storing deletions in a separate area of memory called a **buffer**. The buffer is a fail-safe device. If you erase too much or change your mind, just press CTRL-R to restore the deletion. However, be aware that *SpeedScript* remembers only the last erase or delete you performed.

Another, more powerful use of this buffer is to move or copy sections of text. To move some text from one location in your document to another, first erase or delete it with CTRL-E or CTRL-D. Then move the cursor to where you want the text to appear and press **CTRL-R**. **CTRL-R** instantly inserts the contents of the buffer at the cursor position. If you want to copy some text from one part of your document to another, just erase or delete it with CTRL-E or CTRL-D, restore it at the original position with CTRL-R, then move the cursor elsewhere and press CTRL-R to restore it again. You can retrieve the buffer with CTRL-R as many times as you like. If there is no room left in memory for inserting the buffer, you’ll see the message “Memory Full.”

**Important:** The CTRL-E erase mode lets you erase up to the maximum size of the buffer (2K for disk, about 6K for tape), and CTRL-E also removes the previous contents of the buffer. Keep this in mind if there’s something in the buffer you’d rather keep. If you don’t want the buffer to be erased, hold down the OPTION key while you press CTRL-E. This preserves the buffer contents and adds newly erased text to the buffer.

If you ever need to erase the contents of the buffer, press **CTRL-K** *(kill buffer).*
The Wastebasket Command
If you want to start a new document or simply obliterate all your text, hold down OPTION while you press SHIFT-CLEAR (that's not a combination you're likely to press accidentally). SpeedScript asks, "ERASE ALL TEXT: Are you sure? (Y/N)." This is your last chance. If you don't want to erase the entire document, press N or any other key. Press Y to perform the irreversible deed. There is no way to recover text wiped out with Erase All.

Search and Replace
SpeedScript has a Find command that searches through your document to find a selected word or phrase. A Change option lets you automatically change one word to another throughout the document.

OPTION-CTRL-F (find) activates the search feature,
OPTION-CTRL-C (change) lets you selectively search and replace, and CTRL-G (global) is for automatically searching and replacing.

Searching is a two-step process. First, you need to tell SpeedScript what to search for, then you trigger the actual search. Hold down OPTION and press CTRL-F. The command line prompts "Find:". Type in what you'd like to search for, the search phrase. If you press RETURN alone without typing anything, the Find command is canceled.

When you are ready to search, press CTRL-F. SpeedScript looks for the next occurrence of the search phrase starting from the current cursor position. If you want to hunt through the entire document, press START twice to move the cursor to the very top before beginning the search. Each time you press CTRL-F, SpeedScript looks for the next occurrence of the search phrase and places the cursor at the start of the phrase. If the search fails, you'll see the message "Not Found."

CTRL-C works together with CTRL-F. After you've specified the search phrase with OPTION-CTRL-F, press OPTION-CTRL-C to select the replace phrase. (You can press RETURN alone at the "Change to:" prompt to select a null replace phrase. When you hunt and replace, this deletes the located phrase.) To search and replace manually, start by pressing CTRL-F. After SpeedScript finds the search phrase, press CTRL-C if you want to replace the phrase. If you don't want
to replace the phrase, don’t press CTRL-C. You are not in a special search and replace mode. You’re free to continue writing at any time.

CTRL-G links CTRL-F and CTRL-C together. It first asks “Find:”, then “Change to:”, then automatically searches and replaces throughout the document, starting at the cursor position.

There are a few things to watch out for when using search and replace. First, realize that if you search for the, SpeedScript finds the embedded the in words like therefore and heathen. If you changed all occurrences of the to cow, these words would become cowrefore and heacown. If you want to find a single word, include a space as the first character of the word, since almost all words are preceded by a space. Naturally, if you are replacing, you need to include the space in the replace phrase, too.

SpeedScript also distinguishes between uppercase and lowercase. The word Meldids does not match with meldids. SpeedScript will not find a capitalized word unless you capitalize it in the search phrase. To cover all bases, you will sometimes need to make two passes at replacing a word. Keep these things in mind when using CTRL-G, since you don’t have a chance to stop a global search and replace.

Storing Your Document
Just press CTRL-S to store a document. You’ll see the prompt “Save: (Device:Filename)>”. Type C: for cassette or D: plus a legal Atari filename for disk. If you use the same name as a file already on disk, that file will be replaced by the new one. CTRL-S always saves the entire document. The cursor position within the document is not important.

When the SAVE is complete, SpeedScript reports “No errors” if all is well or gives a message like “Error #144” if not. Check your DOS or BASIC manual for a list of error numbers and their causes.

Loading a Document
To recall a previously saved document, press CTRL-L. Answer the “Load: (Device:Filename)>” prompt with the filename. Again, remember to include the C: for cassette or D: for disk. SpeedScript loads the file and should display “No errors.” Otherwise, SpeedScript reports the error number.
The position of the cursor is important before loading a file. Documents start loading at the cursor position, so be sure to press START twice or OPTION-SHIFT-CLEAR (Erase All) to move the cursor to the start of text, unless you want to merge two documents. When you press CTRL-L to load, the command line turns green to warn you if the cursor is not at the top of the document.

To merge two or more files, simply load the first file, press CTRL-Z to move the cursor to the end of the document, and then load the file you want to merge. Do not place the cursor somewhere in the middle of your document before loading. A load does not insert the text from tape or disk, but overwrites all text after the cursor position. The last character loaded becomes the new end-of-text pointer, and you cannot access any text that appears ahead of this pointer.

Since SpeedScript stores files in ASCII (American Standard Code for Information Interchange), you can load any ASCII file with SpeedScript. You could write a BASIC program with SpeedScript, save it on disk, then use ENTER to read the file from BASIC. In BASIC, you can store a program in ASCII form with LIST "D:filename" for disk or LIST "C:" for tape, ready to load with SpeedScript. You can even load files produced by most other word processors, and most other Atari word processors can read SpeedScript files. You can make full use of SpeedScript’s editing features to prepare ASCII files for the Atari Assembler/Editor, MAC/65, and most other Atari assemblers. And SpeedScript files can be transferred via modem with your favorite telecommunications program that handles ASCII.

**Disk Commands**

Sometimes you forget the name of a file, or need to delete or rename a file. SpeedScript provides a unique mini-DOS for your convenience. Just press **CTRL-M** (menu). SpeedScript reads the entire disk directory and puts it on the screen in three columns. A large cursor shows you which file is currently selected. Use the cursor keys to move the cursor to the file you want to select. A menu at the bottom of the screen shows you what keys you need to press. Press **CTRL-D** to delete the selected file, **R** to rename, **L** to lock, **U** to unlock, or **F** to format the disk. You can load the selected file by pressing **CTRL-L**. The position of the cursor within your document is
not important when loading a file from the menu—*SpeedScript* always erases anything you previously had in memory.

Any changes you make to the directory will not show up until you call up the directory again. Press either 1, 2, 3, or 4 to update the directory from drives 1–4. This also sets the default disk drive, the drive accessed for further changes. When you’re ready to return to writing, press either ESC or the RETURN key.

**Additional Features**

*SpeedScript* has a few commands that don’t do much, but are nice to have. **CTRL-X** exchanges the character under the cursor with the character to the right of the cursor. Thus, you can fix transposition errors with a single keystroke. **CTRL-A** changes the character under the cursor from uppercase to lowercase.

Press **CTRL-B** to change the background and border colors. Each time you press CTRL-B, one of 128 different background colors appears. Press **CTRL-T** (text) to cycle between one of eight text luminances. The colors are preserved until you change them or reboot *SpeedScript*.

If your TV suffers from overscanning, some characters on the left or right margin may be chopped off. Atari *SpeedScript* lets you widen and narrow the width of the screen. Press **OPTION-CTRL-+** (the cursor-left key) to decrease the width of the screen. Each time you press it, the text is reformatted, and the left and right screen margins are adjusted by one character. You can decrease the width all the way down to two characters (although if your screen overscans *that* much, it’s time to buy a new TV). To increase the width, to a maximum of 40 (the default width), press **OPTION-CTRL-*** (the cursor-right key).

One disadvantage of word-wrapping is that it’s hard to tell exactly how many spaces are at the end of a screen line. When a word too long to fit on a line is wrapped to the next line, the hole it left is filled with “false” spaces. That is, the spaces are not actually part of your text and won’t appear on paper. If you want to distinguish between true spaces and false spaces, press **CTRL-O** (on/off). The false spaces become tiny dots. You can write and edit in this mode if you wish, or turn off the feature by pressing CTRL-O again.
Atari SpeedScript disables the BREAK and inverse-video keys when you’re entering or editing text. The inverse-video key was disabled because it is frequently pressed by accident on the 800 and 800XL models. If you want to enter inverse-video characters, hold down SELECT while typing the keys.

Atari 400 and 800 owners will notice that the action of the CAPS/LOWR key has been changed in SpeedScript. It works like the CAPS key on the XL and XE models. Press it once to switch to uppercase, and again to switch to lowercase. In other words, the CAPS/LOWR key toggles between uppercase and lowercase. You can still use SHIFT–CAPS/LOWR to force entry to all uppercase. CTRL–CAPS/LOWR has no effect.

Pressing SYSTEM RESET returns you to SpeedScript without erasing your text when using Atari DOS. With OS/A+ DOS, SYSTEM RESET returns you to the DOS command prompt. You can get back to SpeedScript without losing any text if you type RUN at the prompt.

PRINT!

If you already think SpeedScript has plenty of commands, wait until you see what the printing package offers. SpeedScript supports an array of powerful formatting features. It automatically fits your text between left and right margins which you can specify. You can center a line or block it against the right margin. SpeedScript skips over the perforation on continuous-form paper, or it can wait for you to insert single-sheet paper. A line of text can be printed at the top of each page (a header) and/or at the bottom of each page (a footer), and can include automatic page numbering, starting with whatever number you like. (See page 19 for a summary of the formatting commands.)

SpeedScript can print on different lengths and widths of paper, and single-, double-, triple-, or any-spacing is easy. You can print a document as big as can fit on a tape or disk by linking several files together during printing. You can print to the screen or to a file instead of to a printer. Other features let you send special codes to the printer to control features like underlining, boldfacing, and double-width type (depending on the printer).
But with all this power comes the need to learn additional commands. Fortunately, *SpeedScript* sets most of these variables to a default state. If you don’t change these settings, *SpeedScript* assumes a left margin of 5, a right margin position of 75, no header or footer, single-spacing, and continuous-paper page feeding. You can change these default settings if you want (see below). Before printing, be sure the paper in your printer is adjusted to top-of-form (move the paper perforation just above the printing element). One additional note: Some printers incorporate an automatic skip-over-perforation feature. The printer skips to the next page when it reaches the bottom of a page. Since *SpeedScript* already controls paper feeding, you need to turn off this automatic skip-over-perf feature before running *SpeedScript*, or paging won’t work properly.

To begin printing, simply press **CTRL-P**. *SpeedScript* prompts “Print: (Device:Filename)”. You can print to almost any device, even disk or cassette. If you enter **E** (for Editor), *SpeedScript* prints to the screen, letting you preview where lines and pages break. Enter **P** to Print for most printers. If your printer is attached, powered on, and selected (online), *SpeedScript* begins printing immediately. To cancel printing, hold down the **BREAK** key until printing stops. You can use **CTRL-1** to pause printing. Press **CTRL-1** again to continue.

If you need to print to an RS-232 printer, just Print to a disk file, then boot up your DOS master disk and use the copy selection to copy the print file to the R: device. You can also write BASIC programs to read and process a Printed disk file. Remember, a Print to disk is not the same as a Save to disk.

**Formatting Commands**

The print formatting commands must be distinguished from normal text, so they appear onscreen in inverse video with the text and background colors switched. As mentioned above, the regular inverse-video key is not used for entering inverse-video text. Instead, hold down the **SELECT** key while typing the format key. All lettered printer commands should be entered in lowercase (unSHIFTed). During printing, *SpeedScript* treats these characters as printing commands.

There are two kinds of printing commands, which we’ll call Stage 1 and Stage 2. Stage 1 commands usually control variables such as left margin and right margin. Most are fol-
ollowed by a number, with no space between the command and the number. Stage 1 commands are executed before a line is printed.

Stage 2 commands, like centering and underlining, are executed while the line is being printed. Usually, Stage 1 commands must be on a line of their own, although you can group several Stage 1 commands together on a line. Stage 2 commands are by nature embedded within a line of text. Again, remember to hold down SELECT to enter the boldface characters shown here.

**Stage 1 Commands**

1. **Left margin.** Follow with a number from 0 to 255. Use 0 for no margin. Defaults to 5.
2. **Right margin position,** a number from 1 to 255. Defaults to 75. Be sure the right margin value is greater than the left margin value, or *SpeedScript* will go bonkers.
3. **Top margin.** The position at which the first line of text is printed, relative to the top of the page. Defaults to 5. The header (if any) is always printed on the first line of the page, before the first line of text.
4. **Bottom margin.** The line at which printing stops before continuing to the next page. Standard 8-1/2 × 11 inch paper has 66 lines. Bottom margin defaults to line 58. Don’t make the bottom margin greater than the page length.
5. **Page length.** Defaults to 66. If your printer does not print six lines per inch, multiply lines-per-inch by 11 to get the page length. European paper is usually longer than American paper—11-5/8 or 12 inches. Try a page length of 69 or 72.
6. **Spacing.** Defaults to single-spacing. Follow with a number from 1 to 255. Use 1 for single-spacing, 2 for double-spacing, 3 for triple-spacing.
7. **Start numbering** at page number given. Page numbering normally starts with 1.
8. **Disables printing until selected page number is reached.** For example, a value of 3 would start printing the third page of your document. Normally, *SpeedScript* starts printing with the first page.
9. **Sets the page width,** in columns (think a cross). Defaults to 80. You need to change this for the sake of the centering command if you are printing in double-width or condensed type, or if you are using a 40-column or wide-carriage printer.
Atari SpeedScript 3.0 Keyboard Map

Use **CTRL** with most commands.

- **ESC**: Indent 5 spaces
- **CTRL SET**: Erase: Sentence, Word, Paragraph into buffer
- **CTRL TAB**: Save
- **SHIFT**: Retrieve Buffer
- **ESC I**: Change text luminance
- **CTRL 2**: Insert Mode
- **CTRL 3**: Unused characters
- **CTRL 4**: Show false spaces: ON/OFF
- **CTRL 5**: Erase ALL
- **CTRL 6**: Insert space w/SHIFT: Insert 255 spaces
- **CTRL 7**: Sentence down. w/SHIFT: Next Paragraph
- **CTRL 8**: Delete character. w/SHIFT: Delete spaces.
- **CTRL 9**: Sentence up. w/SHIFT: Paragraph up
- **CTRL 0**: Sentence up. w/SHIFT: Paragraph up
- **CTRL U**: Sentence up. w/SHIFT: Paragraph up
- **CTRL O**: Sentence up. w/SHIFT: Paragraph up

- **CAPS**: Character right. w/SHIFT: Word right
- **SHIFT**: Character left. w/SHIFT: Word left.
  - w/OPTION: Narrow Screen
- **RETURN**: Character right. w/SHIFT: Word right
- **BREAK**: Character right. w/SHIFT: Word right
- **DELETE**: Character right. w/SHIFT: Word right

**RESET**: Forced return to editing mode

**OPTION**: Used with some commands for special option

**SELECT**: Hold down while typing format keys

**START**: Press once: top of screen; twice: top of text
Using *SpeedScript*

### Formatting Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Default</th>
<th>Command</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>b bottom margin</td>
<td>58</td>
<td>p page length</td>
<td>66</td>
</tr>
<tr>
<td>c centering</td>
<td></td>
<td>f right margin</td>
<td>75</td>
</tr>
<tr>
<td>e edge right</td>
<td></td>
<td>g spacing</td>
<td>1</td>
</tr>
<tr>
<td>f define footer</td>
<td></td>
<td>h top margin</td>
<td>5</td>
</tr>
<tr>
<td>g goto linked file</td>
<td></td>
<td>i underline toggle</td>
<td></td>
</tr>
<tr>
<td>h define header</td>
<td></td>
<td>j page wait</td>
<td>off</td>
</tr>
<tr>
<td>i information</td>
<td></td>
<td>x columns across</td>
<td>80</td>
</tr>
<tr>
<td>j select linefeeds</td>
<td></td>
<td>h page number</td>
<td></td>
</tr>
<tr>
<td>l left margin</td>
<td>5</td>
<td>@ starting page number</td>
<td>1</td>
</tr>
<tr>
<td>m margin release</td>
<td></td>
<td>? print starting with #</td>
<td>1</td>
</tr>
<tr>
<td>n next page</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**n** Forced paging. Normally, *SpeedScript* prints the footer and moves on to the next page only when it has finished a page, but you can force it to continue to the next page by issuing this command. It requires no numbers.

**m** Margin release. Disables the left margin for the next printed line. Remember that this executes before the line is printed. It’s used for outdenting.

**w** Page wait. This command should be placed at the beginning of your document before any text. With page wait turned on, *SpeedScript* prompts you to “Insert next sheet, press RETURN” when each page is finished printing. Insert the next sheet, line it up with the printhead, then press RETURN to continue. Page wait is ignored during disk or screen output.

**j** Select automatic linefeeds after carriage return. Like **w**, this command must be placed before any text. Don’t use this command to achieve double-spacing, but only if all text prints on the same line.

**i** Information. This works like REM in BASIC. You follow the command with a line of text, up to 255 characters, ending in a return-mark. This line will be ignored during printing and is handy for making such notes to yourself as the filename of the document.
h Header define and enable. The header must be a single line of text (up to 254 characters) ending in a return-mark. The header prints on the first line of each page. You can include Stage 2 commands such as centering and page numbering in a header. You can use a header by itself without a footer. The header and footer should be defined at the top of your document, before any text. If you want to prevent the header from printing on the first page, put a return-mark by itself at the top of your document before the header definition.

f Footer define and enable. The footer must be a single line of text (up to 254 characters) ending in a return-mark. The footer prints two lines prior to the last line of each page. As with the header, you can include Stage 2 printing commands, and you don’t need to set the header to use a footer.

g Go to (link) next file. Put this command as the last line in your document. Follow the command with the filename, including D: for disk. After the text in memory is printed, the link command loads the next file into memory. You can continue linking in successive files, but don’t include a link in the last file. Before you start printing a linked file, make sure the first of the linked files is in memory. When printing is finished, the last file linked to will be in memory.

**Stage 2 Commands**

These commands either precede a line of text or are embedded within one.

c Centering. Put this at the beginning of a line you want to center. This will center only one line ending in a return-mark. Repeat this command at the beginning of every line you want centered. Centering uses the page-width setting (see above) to center the line properly. To center a double-width line, either set the page width to 40 or pad out the rest of the line with an equal number of spaces. If you use double-width, remember that the spaces preceding the centered text will be double-wide spaces.

e Edge right. This works in the same manner as centering, but it blocks the line flush with the right margin.

# When *SpeedScript* encounters this command, it prints the current page number. You usually embed this within a header or footer.
A simple form of underlining. It works only on printers that recognize CHR$(8) as a backspace and CHR$(95) as an underline character. Underlining works on spaces, too. Use the first u to start underlining and another one to turn off underlining.

**Fonts and Styles**

Most dot-matrix printers are capable of more than just printing text at ten characters per inch. Some printers have several character sets, with italics and foreign language characters. Most can print in double-width (40 characters per line), condensed (132 characters per line), and in either pica or elite. Other features include programmable characters, programmable tab stops, and graphics modes. Many word processors customize themselves to a particular printer, but *SpeedScript* was purposely designed not to be printer-specific. Instead, *SpeedScript* lets you define your own Stage 2 printing commands.

You define a programmable printkey by choosing any character that is not already used for other printer commands. The entire uppercase alphabet is available for printkeys, and you can choose letters that are related to their function (like D for double-width). You enter these commands like printer commands, by holding down SELECT while you type them. The printkeys are like variables in BASIC.

To define a printkey, just hold down SELECT while you type the key you want to assign as the printkey, then an equal sign (=), and finally the ASCII value to be substituted for the printkey during printing. Now, whenever *SpeedScript* encounters the printkey embedded in text, it prints the character with the ASCII value you previously defined.

For example, to define the + key as the letter z, you first look up the ASCII value of z (in either your printer manual or in any Atari manual). The ASCII value of the letter z is 122, so the definition is

```
G-122
```

Now, anywhere you want to print the letter z, substitute the printkey:

```
Gadooks! The Zoo is Many!
```
This would appear on paper as

**Gadzooks! The zoo is zany!**

More practically, here's how you could program italics on an Epson MX-80-compatible printer. You switch on italics by sending an ESC (a character with an ASCII value of 27), then the character 4. You turn off italics by sending ESC 5. So define \texttt{SHIFT-E} as the escape code. Anywhere you want to print a word in italics, bracket it with printkey \texttt{E}, then 4, and printkey \texttt{E}, then 5:

\begin{quote}
The word \texttt{4italics35} is in italics.
\end{quote}

You can similarly define whatever codes your printer uses for features like double-width or emphasized mode. For your convenience, four of the printkeys are predefined, though you can change them. Keys 1-4 are defined as 27, 14, 15, and 18, common values for most printers. On most printers, \texttt{CHR$(27)$} is the ESCape key, \texttt{CHR$(14)$} starts double-width, \texttt{CHR$(15)$} either stops double-width or starts condensed characters, and \texttt{CHR$(18)$} usually cancels condensed characters.

\texttt{SpeedScript} actually lets you embed any character within text, so you may prefer to put in the actual printer codes as part of your text. To set italics, you could just press ESC twice, then 4. The ESC key appears in text as a mutant \texttt{E}. Double-width has a value of 14, the same value as CTRL-N. To start double-width, just embed a CTRL-N. Remember that you must press ESC before any CTRL key to get it to appear in text. CTRL keys appear as small "shadowed" capital letters. These characters, though, are counted as part of the length of a line, and excessive use within one line can result in a shorter than normal line. It can be more convenient to use the printkeys, since if you ever change printers, you have to change only the definitions of the keys.

Keep one thing in mind about printkeys: \texttt{SpeedScript} always assumes it is printing to a rather dumb, featureless printer, the least common denominator. \texttt{SpeedScript} doesn't understand the intent of a printkey; it justs sends out its value. So if you make one word within a line double-width, it may make the line overflow the specified right margin. There's no
way for *SpeedScript* to include built-in font and typestyle codes without being customized for a particular printer since no set of codes is universal to all printers.

**Hints and Tips**

It may take you awhile to fully master *SpeedScript*, but as you do, you’ll discover many ways to use the editing and formatting commands. For example, there is a simple way to simulate tab stops, say, for a columnar table. Just type a period at every tab stop position. Erase the line with CTRL-E, then restore it with CTRL-R multiple times. When you are filling in the table, just use word left/word right to jump quickly between the periods. Or you can use the programmable printkeys to embed your printer’s own commands for setting and jumping to tab stops.

You don’t have to change or define printer commands every time you write. Just save these definitions and load this file each time you write. You can create many custom definition files and have them ready to use on disk. You can create customized “fill-in-the-blank” letters. Just type the letter, and everywhere you’ll need to insert something, substitute a unique character, such as an * or a CTRL character. When you’re ready to customize the letter, use Find to locate each symbol and insert the specific information. Instead of typing an oft-used word or phrase, substitute a unique character, then use CTRL-G to globally change these characters into the actual word or phrase. You can even use *SpeedScript* as a simple filing program. Just type in all your data, flagging each field with a unique character. You can use Find to quickly locate any field.
Chapter 2
Entering SpeedScript
The Machine Language Editor: MLX

Two program-entry aids written in BASIC are included here to make typing in SpeedScript as easy as possible. The first, "MLX," is explained in this article. The second, "The Automatic Proofreader," is a short program that will help you type in MLX without typing mistakes. Read the instructions for using the Automatic Proofreader later in this chapter before you type in the MLX program.

"MLX" is a new way to enter long machine language (ML) programs with a minimum of fuss. MLX lets you enter the numbers from a special list that looks similar to BASIC DATA statements. It checks your typing on a line-by-line basis. It won't let you enter illegal characters when you should be typing numbers. It won't let you enter numbers greater than 255 (forbidden in ML). And it won't let you enter the wrong numbers on the wrong line. In addition, MLX creates a ready-to-use tape or disk file.

Using MLX

Type in and save MLX, Program 2-1 (you'll want to use it in the future). When you're ready to type in SpeedScript, run MLX. MLX asks you for three numbers: the starting address, the ending address, and the run/init address. These numbers for SpeedScript are

Starting Address? 7936
Ending Address? 16229
Run/Init Address 7936

Next, you'll be asked "Tape or Disk." SpeedScript can be saved as either a binary file on disk or as a boot tape. Press T for use with a tape drive. If you press D for disk, you'll be asked "Boot Disk or Binary File." Press F to select the Binary File option. Although you could save SpeedScript as an auto-booting disk, it makes no sense, since such a disk cannot contain DOS, which is necessary for file-oriented disk access.

The screen will then show the first prompt, the number 7936 followed by a colon. Type in each three-digit number
shown in the listing. You do not need to type the comma shown in the listing; MLX inserts the comma automatically. The prompt is the current line you are entering from the listing. It increases by six each time you enter a line. That’s because each line has seven numbers—six actual data numbers plus a checksum number. The checksum verifies that you typed the previous six numbers correctly. If you enter any of the six numbers wrong, or if you enter the checksum wrong, the computer rings a buzzer and prompts you to reenter the line. If you enter it correctly, a bell tone sounds and you continue to the next line.

MLX accepts only numbers as input. If you make a typing error, press the DELETE/BACK S key; the entire number is deleted. You can press it as many times as necessary back to the start of the line. If you enter three-digit numbers as listed, the computer automatically prints the comma and goes on to accept the next number. If you enter less than three digits, you can press the comma key, the space bar, or the RETURN key to advance to the next number. The checksum automatically appears in inverse video for emphasis.

**MLX Commands**

When you finish typing an ML listing (assuming you type it all in one session), you can then save the completed program on tape or disk. Follow the screen instructions. If you get any errors while saving, you probably have a bad disk or the disk is full or you made a typo when entering the MLX program itself.

Fortunately, you don’t have to enter all of *SpeedScript* in one sitting. MLX lets you enter as much as you want, save it, and then reload the file from tape or disk later. MLX recognizes these commands:

- **CTRL-S**  Save
- **CTRL-L**  Load
- **CTRL-N**  New Address
- **CTRL-D**  Display

To issue a command, hold down the CTRL key (CONTROL on the XL models) and press the indicated key. When you enter a command, MLX jumps out of the line you’ve been typing, so we recommend you do it at a new prompt. Use the Save command (CTRL-S) to save what you’ve been working
on. It will save on tape or disk as if you've finished, but the
tape or disk won't work, of course, until you finish the typing.
Remember to make a note of the address where you stop. The
next time you run MLX, answer all the prompts as you did
before—regardless of where you stopped typing—then insert
the disk or tape. When you get to the line number prompt,
press CTRL-L to reload the partly completed file into memory.
Then use the New Address command to resume typing.

To use the New Address command, press CTRL-N and
enter the address where you previously stopped. The prompt
will change, and you can then continue typing. Always enter a
New Address that matches up with one of the line numbers in
the MLX-format listing, or the checksum won't work. The Dis-
play command lets you display a section of your typing. After
you press CTRL-D, enter two addresses within the line-number
range of the listing. You can break out of the listing display
and return to the prompt by pressing any key.

Program 2-1. MLX: The Machine Language Editor
Refer to the "Automatic Proofreader" article before typing in this program.

DA 100 GRAPHICS 0: DL=PEEK(560)+256*PEEK(561)+4
POKE DL-1,71:POKE DL+2,6
NJ 110 POSITION 8,0:"MLX":POSITION 23,0:"Tail safe entry":POKE 710,0:?
JK 120 ? "Starting Address":INPUT BEG: ? "Ending Address":INPUT FIN: ? "Run/Init Address":INPUT STARTADR
DD 130 DIM A(6),BUFFERS(FIN-BEG+127),T$(20),F$(20),CIO$(7),SECTOR$(128),DSKINV$(6)
JJ 140 OPEN #1,4,0,"K:" ?: ?: "Tape or Disk:";
BM 150 BUFFERS=CHR$(0):BUFFERS(FIN-BEG+30)=BUFFERS(2)=BUFFERS(SECTOR$:BUFFERS$=BUFFERS$=BUFFERS$(FIN-BEG+127)/128)
GC 160 ADDR=BEG:CIO$="huh":CIO$(4)=CHR$(170):CIO$(5)="LV":CIO$(7)=CHR$(228)
EJ 170 GET #1,MEDIA:IF MEDIA<84 AND MEDIA<68 THEN 170
PO 180 ? CHR$(MEDIA): ? :IF MEDIA<ASC("T") THEN BUFFER$="":GOTO 250
PL 190 BEG=BEG-24:BUFFERS=CHR$(0):BUFFERS(2)=CHR$(INT((FIN-BEG+127)/128))
KF 200 H=INT(BEG/256):L=BEG-H*256:BUFFERS(3)=CHR$(L):BUFFERS(4)=CHR$(H)
EC 210 PINIT=BEG+8:H=INT(PINIT/256):L=PINIT-H*256:BUFFERS(5)=CHR$(L):BUFFERS(6)=CHR$(H)
PB 220 FOR I=7 TO 24:READ A:BUFFER$(I)=CHR$(A)
: NEXT I:DATA 24,96,169,60,141,2,211,169
,0,133,10,169,0,133,11,76,0,0
DP 230 H=INT(STARTADR/256):L=STARTADR-H*256:BU
FFER$(15)=CHR$(L):BUFFER$(19)=CHR$(H)
KL 240 BUFFER$(23)=CHR$(L):BUFFER$(24)=CHR$(H)
HI 250 IF MEDIA<ASC("D") THEN 360
DO 260 ? "Boot Disk or Binary File:";
LI 270 GET #1,DTYPE:IF DTYPE<>68 AND DTYPE<>70
 THEN 270
GM 280 ? CHR$(DTYPE):IF DTYPE=70 THEN 360
PJ 290 BEG=BEG-30:BUFFER$=CHR$(0):BUFFER$(2)=C
HR$(INT((FIN-BEG+127)/128))
KG 300 H=INT(BEG/256):L=BEG-H*256:BUFFER$(3)=C
HR$(L):BUFFER$(4)=CHR$(H)
HH 310 PINIT=STARTADR:H=INT(PINIT/256):L=PINIT
-H*256:BUFFER$(5)=CHR$(L):BUFFER$(6)=CHR$(H)
AO 320 RESTORE 330:FOR I=7 TO 30:READ A:BUFFER
$(I)=CHR$(A):NEXT I
GA 330 DATA 169,0,141,231,2,133,14,169,0,141,2
32,2,133,15,169,0,133,10,169,0,133,11,2
4,96
DB 340 H=INT(BEG/256):L=BEG-H*256:BUFFER$(8)=C
HR$(L):BUFFER$(15)=CHR$(H)
DO 350 H=INT(STARTADR/256):L=STARTADR-H*256:BU
FFER$(22)=CHR$(L):BUFFER$(26)=CHR$(H)
JP 360 GRAPHICS 0:POKE 712,10:POKE 710,10:POKE
709,2
JK 370 ? ADDR:"":FOR J=1 TO 6
NF 380 GOSUB 570:IF N=-1 THEN J=J-1:GOTO 380
BF 390 IF N=-19 THEN 720
DI 400 IF N=-12 THEN LET READ=1:GOTO 720
AJ 410 TRAP 410:IF N=-14 THEN ?:?:"New Addres
s"::INPUT ADDR:??:GOTO 370
HO 420 TRAP 40000:IF N<>-4 THEN 480
AJ 430 TRAP 430:?:?:"Display:From"::INPUT F:?:
"To":,:INPUT T:TRAP 32767
ML 440 IF F<BEG OR F>FIN OR T<BEG OR T>FIN OR
T<F THEN ?CHR$(253);"At least ";BEG;",No
Not More Than ";FIN:GOTO 430
MH 450 FOR I=F TO T STEP 6:?:?:I;":":FOR K=0
TO 5:N=PEEK(ADR(BUFFER$)+I+K-BEG):T$="
\000":T$$(4-LEN STR$(N))=STR$(N)
MA 460 IF PEEK(764)<255 THEN GET #1,A:POP :POP
?:?:GOTO 370
FM 470 ? T$":":NEXT K:?CHR$(126);:NEXT I:?:
?:?:GOTO 370
GA 480 IF N<0 THEN ?:GOTO 370
MH 490 A(J)=N:NEXT J
Entering SpeedScript

JM 500  CKSUM=ADDR-INT(ADDR/256)*256:FOR I=1 TO 6:CKSUM=CKSUM+A(I):CKSUM=CKSUM-256*(CKSUM>255):NEXT I
KK 510  RF=128:SOUND 0,200,12,8:GOSUB 570:SOUND 0,0,0,RF=0:?CHR$(126)
CN 520  IF N<CKSUM THEN ? :"Incorrect";CHR$(253):?:GOTO 370
EK 530  FOR W=15 TO 0 STEP -1:SOUND 0,50,10,W:NEXT W
FL 540  FOR I=1 TO 6:POKE ADR(BUFFER$)+ADDR-BEG+I-1,A(I):NEXT I
HB 550  ADDR=ADDR+6:IF ADDR<=FIN THEN 370
SM 560  GOTO 710
FI 570  N=0;Z=0
PH 580  GET #1,A:IF A=155 OR A=44 OR A=32 THEN 670
FB 590  IF A<32 THEN N=-A:RETURN
EB 600  IF A<126 THEN 630
ML 610  GOSUB 690:IF I=1 AND T=44 THEN N=-1:?CHR$(126);:GOTO 690
SN 620  GOTO 570
GI 630  IF A<48 OR A>57 THEN 580
AN 640  ?CHR$(A+RF);:N=N*10+A-48
EB 650  IF N>255 THEN ?CHR$(253);:A=126;GOTO 600
EH 660  Z=Z+1:IF Z<3 THEN 580
JH 670  IF Z=0 THEN ?CHR$(253);:GOTO 570
KC 680  ?",":RETURN
NO 690  POKE 752,1:FOR I=1 TO 3:?CHR$(30);:GET #6,T:IF T<44 AND T<58 THEN ?CHR$(A);:NEXT I
PI 700  POKE 752,0:?"";CHR$(126);:RETURN
KM 710  GRAPHICS 0:POKE 710,26:POKE 712,26:POKE 709,2
FF 720  IF MEDIA=ASC("T") THEN 890
DJ 730  REM DISK
DK 740  IF READ THEN ? :? "Load File":?
IG 750  IF DTYPE<70 THEN 1040
AE 760  ?:?:"Enter AUTORUN.SYS for automatic use":?:"Enter filename":INPUT T$;GF 770  F$=T$:IF LEN(T$)>2 THEN IF T$(1,2)<"D:" THEN F$="D:";F$(3)=T$
NJ 780  TRAP 870:CLOSE #2:OPEN #2,8-4:READ,0,F$:?:?:"Working..."
JM 790  IF READ THEN FOR I=1 TO 6:GET #2,A:NEXT I:GOTO 820
PO 800  PUT #2,255:PUT #2,255
NF 820 GOSUB 970: IF PEEK(195) > 1 THEN 870
IF 830 IF STARTADR = 0 OR READ THEN 850
FD 840 PUT #2, 224: PUT #2, 2: PUT #2, 225: PUT #2, 2
: H = INT(STARTADR / 256): L = STARTADR - H * 256: PUT #2, L: PUT #2, H
GC 850 TRAP 40000: CLOSE #2: ? "Finished."
: IF READ THEN ? ? LET READ = 0: GOTO 360
HF 860 END
FD 870 ? "Error "; PEEK(195);" trying to access"
MC 880 REM "BOOT TAPE"
HN 890 IF READ THEN ? ? "Read Tape"
LP 910 ? ? "Press RETURN when ready: "
JH 920 TRAP 960: CLOSE #2: OPEN #2, B-4*READ, 128,
: "C: "; ? ? "Working..."
NH 930 GOSUB 970: IF PEEK(195) > 1 THEN 960
GC 940 CLOSE #2: TRAP 40000: ? "Finished.": ? ?: IF READ THEN LET READ = 0: GOTO 360
HF 950 END
CD 960 ? ?: "Error "; PEEK(195);" when reading/ writing boot tape"?: CLOSE #2: GOTO 890
MB 970 REM "CIO Load/Save File#2 opened READ=0"
for write, READ=1 for read
EA 980 X = 32: REM File#2, $20
EF 990 ICCOM = 834: ICBADR = 836: ICBLEN = 840: ICSTAT = 835
MD 1000 H = INT(ADR(BUFFER$) / 256): L = ADR(BUFFER$) - H * 256: POKE ICBADR + X, L: POKE ICBADR + X + 1,
: H
FH 1010 L = FIN - BEG + 1: H = INT(L / 256): L = L - H * 256: POKE ICBLEN + X, L: POKE ICBLEN + X + 1, H
MD 1020 POKE ICCOM + X, 11 - 4 * READ: A = USR(ADR(CIO$), X)
BG 1030 POKE 195, PEEK(ICSTAT): RETURN
KA 1040 REM "SECTOR I/O"
GC 1050 IF READ THEN 1100
HE 1060 ? ?: "Format Disk In Drive 1? (Y/N): "
PC 1070 GET #1, A: IF A > 78 AND A < 9 THEN 1070
EC 1080 ? CHR$(A): IF A = 78 THEN 1100
CP 1090 ? ?: "Formatting...": XIO 254, #2, 0, 0, "D"
: ?: "Format Complete":?
AC 1100 NR = INT((FIN-BEG + 127) / 128): BUFFER$(FIN-BEG + 2) = CHR$(0): IF READ THEN ? "Reading...": GOTO 1120
LE 1110 ? "Writing...
LI 1120 FOR I = 1 TO NR: S = I
IO 1130 IF READ THEN GOSUB 1220: BUFFER$(I * 128 - 127) = SECTOR$: GOTO 1160
Entering SpeedScript

PL 1140 SECTOR$=BUFFER$(I*128-127)
AM 1150 GOSUB 1220
DN 1160 IF PEEK(DSTATS)<<1 THEN 1200
FB 1170 NEXT I
GM 1180 IF NOT READ THEN END
DH 1190 ? :? :LET READ=0:GOTO 360
JJ 1200 ? "Error on disk access." :? "May need formatting." :GOTO 1040
KI 1210 REM
BL 1220 REM SECTOR ACCESS SUBROUTINE
IG 1230 REM Drive ONE
IH 1240 REM Pass buffer in SECTOR$
MP 1250 REM sector # in variable S
EG 1260 REM READ=1 for read,
KJ 1270 REM READ=0 for write
BN 1280 BASE=3*256
GL 1290 DUNIT=BASE+1:DCOMND=BASE+2:DSTATS=BASE+3
NL 1300 DBUFLO=BASE+4:DBUFHI=BASE+5
AI 1310 DBYTL=BASE+8:DBYTHI=BASE+9
JA 1320 DAUX1=BASE+10:DAUX2=BASE+11
PN 1330 REM DIM DSKINV$(4)
CA 1340 DSKINV$="hLS":DSKINV$(4)=CHR$(228)
BP 1360 POKE DBUFHI,H
CO 1370 POKE DBUFLO,L
PD 1380 POKE DCOMND,87-5*READ
AA 1390 POKE DAUX2,INT(S/256):POKE DAUX1,S-PEEK(DAUX2)*256
KJ 1400 A=USR(ADR(DSKINV$))
KG 1410 RETURN

33
The Automatic Proofreader

At last there's a way for your computer to help you check your typing. "The Automatic Proofreader" will make entering programs faster, easier, and more accurate.

The strong point of computers is that they excel at tedious, exacting tasks. So why not get your computer to check your typing for you?

"The Automatic Proofreader" will help you type in "MLX" program listings without typing mistakes. It is a short error-checking program that hides itself in memory. When activated, it lets you know immediately after typing a line from a program listing if you have made a mistake. Please read these instructions carefully before typing the MLX program.

Preparing the Proofreader

1. Type in the Proofreader (Program 2-2). Be very careful when entering the DATA statements—don't type an l instead of a 1, an O instead of a 0, extra commas, and so on.
2. Save the Proofreader on tape or disk at least twice before running it for the first time.
3. After the Proofreader is saved, type RUN. It will check itself for typing errors in the DATA statements and warn you if there's a mistake. Correct any errors and save the corrected version. Keep a copy in a safe place—you'll need it again and again when typing in programs from other COMPUTE! books or COMPUTE! magazine.
4. When a correct version of the Proofreader is run, the following message will appear on the screen: "Automatic Proofreader Now Activated." Type NEW and press RETURN. You are now ready to enter the MLX program listing. If you press SYSTEM RESET, the Proofreader is disabled. To reactivate it, just type PRINT USR(1536) and press RETURN.
Using the Proofreader

The MLX program listing has a checksum found immediately to the left of each line number. Don't enter the checksum when typing in a program. It is just for your information.

When you type in a line from the program listing and press RETURN, the Proofreader displays the checksum letters at the top of your screen. These checksum letters must match the checksum letters in the printed listing. If they don’t match, it means you typed the line differently from the way it is listed. Immediately recheck your typing. You can correct any mistakes you find.

The Proofreader is not picky with spaces. It will not notice extra spaces or missing ones. This is for your convenience since spacing is generally not important. But occasionally proper spacing is important, so be extra careful with spaces. The Proofreader will catch practically everything else that can go wrong. Characters in inverse video will appear like this:

```
INVERSE VIDEO
```

Enter these characters with the Atari key.

Due to the nature of a checksum, the Proofreader will not catch all errors. The Proofreader will not catch errors of transposition. In fact, you could type in a line in any order, and the Proofreader wouldn’t notice.

There’s another thing to watch out for: If you enter a line by using abbreviations for commands, the checksum will not match up. But there is a way to make the Proofreader check the line. After entering the line, LIST it. This eliminates the abbreviations. Then move the cursor up to the line and press RETURN. It should now match the checksum. You can check whole groups of lines this way. The only abbreviation that cannot be handled this way is when a question mark (?) is used instead of PRINT; they are not the same to the Proofreader.

Program 2-2. The Automatic Proofreader

```
100  GRAPHICS 0
110  FOR I=1536 TO 1700:READ A:POKE I,A:CK=C
     K+A:NEXT I
120  IF CK<>1972 THEN ? "ERROR IN DATA STATEMENTS. CHECK TYPING."
     :END
130  A=USR(1536)
```
"AUTOMATIC PROOFREADER NOW ACTIVATED."

```
140   DATA 104, 160, 0, 185, 26, 3
1536  DATA 201, 69, 240, 7, 200, 200
1548  DATA 192, 34, 208, 243, 96, 200
1554  DATA 169, 74, 153, 26, 3, 200
1560  DATA 169, 6, 153, 26, 3, 162
1566  DATA 0, 189, 0, 228, 157, 74
1572  DATA 6, 232, 224, 16, 208, 245
1578  DATA 169, 93, 141, 78, 6, 169
1584  DATA 6, 141, 79, 6, 24, 173
1590  DATA 4228, 105, 1, 141, 95
1596  DATA 6, 173, 5, 228, 105, 0
1602  DATA 141, 96, 6, 169, 0, 133
1608  DATA 203, 96, 247, 238, 125, 241
1614  DATA 93, 6, 244, 241, 115, 241
1620  DATA 124, 241, 76, 205, 238, 0
1626  DATA 0, 0, 0, 0, 32, 62
1632  DATA 246, 8, 201, 155, 240, 13
1638  DATA 201, 32, 240, 7, 72, 24
1644  DATA 101, 203, 133, 203, 104, 40
1650  DATA 96, 72, 152, 72, 138, 72
1656  DATA 160, 0, 169, 128, 145, 88
1662  DATA 200, 192, 40, 208, 249, 165
1668  DATA 203, 74, 74, 74, 74, 24
1674  DATA 105, 161, 160, 3, 145, 88
1680  DATA 165, 203, 41, 15, 24, 105
1686  DATA 161, 200, 145, 88, 169, 0
1692  DATA 133, 203, 104, 170, 104, 168
1698  DATA 104, 40, 96
```
Before you begin typing *SpeedScript*, you must load and run the "MLX" program. Answer the MLX prompts as follows:

- **Starting Address?** 7936
- **Ending Address?** 16229
- **Run/Init Address** 7936

### Program 2-3. SpeedScript

To enter this program, you must use Program 2-1, MLX, found earlier in this chapter.

<table>
<thead>
<tr>
<th>Address</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>7936:</td>
<td>173, 198, 002, 141, 197, 002, 201</td>
</tr>
<tr>
<td>7942:</td>
<td>032, 137, 037, 169, 203, 205, 021</td>
</tr>
<tr>
<td>7948:</td>
<td>179, 066, 141, 179, 066, 240, 115</td>
</tr>
<tr>
<td>7954:</td>
<td>033, 032, 031, 037, 032, 080, 007</td>
</tr>
<tr>
<td>7960:</td>
<td>042, 165, 012, 141, 118, 037, 027</td>
</tr>
<tr>
<td>7966:</td>
<td>165, 013, 141, 119, 037, 169, 162</td>
</tr>
<tr>
<td>7972:</td>
<td>117, 133, 012, 169, 037, 133, 125</td>
</tr>
<tr>
<td>7978:</td>
<td>013, 169, 000, 141, 068, 002, 179</td>
</tr>
<tr>
<td>7984:</td>
<td>169, 001, 133, 009, 032, 234, 114</td>
</tr>
<tr>
<td>7990:</td>
<td>037, 076, 072, 038, 000, 000, 021</td>
</tr>
<tr>
<td>7996:</td>
<td>000, 000, 000, 000, 000, 000, 060</td>
</tr>
<tr>
<td>8002:</td>
<td>000, 000, 000, 000, 000, 000, 066</td>
</tr>
<tr>
<td>8008:</td>
<td>000, 000, 000, 000, 000, 000, 072</td>
</tr>
<tr>
<td>8014:</td>
<td>000, 000, 000, 000, 000, 000, 078</td>
</tr>
<tr>
<td>8020:</td>
<td>000, 000, 000, 000, 000, 000, 084</td>
</tr>
<tr>
<td>8026:</td>
<td>000, 000, 000, 000, 000, 000, 090</td>
</tr>
<tr>
<td>8032:</td>
<td>000, 000, 000, 000, 000, 000, 096</td>
</tr>
<tr>
<td>8038:</td>
<td>000, 000, 000, 000, 000, 000, 102</td>
</tr>
<tr>
<td>8044:</td>
<td>000, 000, 000, 000, 000, 000, 108</td>
</tr>
<tr>
<td>8050:</td>
<td>000, 000, 000, 000, 000, 000, 114</td>
</tr>
<tr>
<td>8056:</td>
<td>000, 000, 000, 000, 000, 000, 120</td>
</tr>
<tr>
<td>8062:</td>
<td>000, 000, 000, 000, 000, 000, 126</td>
</tr>
<tr>
<td>8068:</td>
<td>000, 000, 000, 000, 000, 000, 132</td>
</tr>
<tr>
<td>8074:</td>
<td>000, 000, 000, 000, 000, 000, 138</td>
</tr>
<tr>
<td>8080:</td>
<td>000, 000, 000, 000, 000, 000, 144</td>
</tr>
<tr>
<td>8086:</td>
<td>000, 000, 000, 000, 000, 000, 150</td>
</tr>
<tr>
<td>8092:</td>
<td>000, 000, 000, 000, 000, 000, 156</td>
</tr>
<tr>
<td>8098:</td>
<td>000, 000, 000, 000, 000, 000, 162</td>
</tr>
<tr>
<td>8104:</td>
<td>000, 000, 000, 000, 000, 000, 168</td>
</tr>
<tr>
<td>8110:</td>
<td>000, 000, 000, 000, 000, 000, 174</td>
</tr>
<tr>
<td>8116:</td>
<td>000, 000, 000, 000, 000, 000, 180</td>
</tr>
<tr>
<td>8122:</td>
<td>000, 000, 000, 000, 000, 000, 186</td>
</tr>
<tr>
<td>8128:</td>
<td>000, 000, 000, 000, 000, 000, 192</td>
</tr>
</tbody>
</table>
8134:000,000,000,000,000,000,198
8140:000,000,000,000,000,000,204
8146:000,000,000,000,000,000,210
8152:000,000,000,000,000,000,216
8158:000,000,000,000,000,000,222
8164:000,000,000,000,000,000,228
8170:000,000,000,000,000,000,234
8176:000,000,000,000,000,000,240
8182:000,000,000,000,000,000,246
8188:036,037,045,017,000,000,131
8194:000,000,000,000,000,000,002
8200:024,024,024,024,024,024,128
8206:000,024,000,102,102,102,088
8212:000,000,000,000,102,122
8218:102,102,255,102,000,074
8224:024,062,096,060,066,124,148
8230:024,000,000,204,216,048,018
8236:096,204,140,000,000,056,028
8242:108,056,112,222,204,118,102
8248:000,024,024,048,000,000,152
8254:000,000,000,024,048,096,230
8260:096,096,048,024,000,048,124
8266:024,012,012,012,024,048,206
8272:000,000,102,060,255,060,045
8278:102,000,000,000,024,024,236
8284:126,024,024,000,000,000,010
8290:000,000,000,048,048,096,034
8296:000,000,000,000,126,000,230
8302:000,000,000,000,000,000,110
8308:000,000,000,000,000,000,212
8314:006,012,024,048,096,192,244
8320:000,124,026,222,246,230,132
8326:198,124,000,024,056,024,048
8332:024,024,024,126,000,124,026
8338:198,012,024,048,096,254,010
8344:000,254,012,024,056,012,254
8350:198,124,000,028,060,108,164
8356:204,254,012,012,000,254,132
8362:192,252,006,006,198,124,180
8368:000,124,192,252,198,198,116
8374:198,124,000,126,006,012,136
8380:024,048,096,096,000,124,064
8386:198,198,124,198,198,124,210
8392:000,124,198,198,126,012,090
8398:024,048,000,000,048,048,118
8404:000,048,048,000,000,000,052
8410:048,048,000,048,048,096,250
8416:000,012,024,048,096,048,196
8422:024,012,000,000,126,136
8428:000,000,126,000,000,048,154

SpeedScript
Entering SpeedScript

8434: 024, 012, 006, 012, 024, 048, 112
8440: 000, 060, 102, 006, 012, 024, 048
8446: 000, 024, 000, 124, 198, 222, 054
8452: 214, 220, 224, 060, 000, 124, 078
8458: 198, 198, 198, 254, 198, 198, 230
8464: 000, 252, 198, 198, 252, 198, 090
8470: 198, 252, 000, 124, 198, 192, 218
8476: 192, 192, 198, 124, 000, 248, 214
8482: 204, 198, 198, 204, 248, 004
8488: 000, 254, 192, 192, 252, 192, 098
8494: 192, 254, 000, 254, 192, 192, 106
8500: 252, 192, 192, 192, 000, 124, 236
8506: 198, 192, 222, 198, 198, 124, 166
8512: 000, 198, 198, 198, 254, 198, 086
8518: 198, 198, 206, 126, 024, 024, 128
8524: 024, 024, 126, 000, 062, 080
8530: 012, 012, 012, 204, 120, 198
8536: 000, 198, 204, 216, 240, 216, 138
8542: 204, 198, 000, 192, 192, 192, 048
8548: 192, 192, 192, 254, 000, 198, 104
8554: 238, 254, 214, 198, 198, 198, 126
8560: 000, 198, 230, 246, 254, 222, 238
8566: 206, 198, 000, 124, 198, 198, 018
8572: 198, 198, 198, 124, 000, 252, 070
8578: 198, 198, 198, 252, 192, 192, 080
8584: 124, 198, 198, 198, 222, 052
8590: 124, 014, 000, 252, 198, 198, 160
8596: 252, 216, 204, 198, 000, 124, 118
8602: 198, 192, 124, 006, 198, 124, 228
8608: 000, 126, 024, 024, 024, 024, 126
8614: 024, 024, 000, 198, 198, 198, 040
8620: 000, 198, 198, 198, 124, 000, 198, 064
8626: 198, 198, 198, 198, 108, 056, 110
8632: 000, 198, 198, 198, 214, 254, 222
8638: 238, 198, 000, 198, 198, 108, 106
8644: 056, 108, 198, 198, 000, 102, 090
8650: 102, 102, 060, 024, 024, 024, 026
8656: 000, 254, 012, 024, 048, 096, 130
8662: 192, 254, 000, 030, 024, 024, 226
8668: 024, 024, 024, 030, 000, 064, 130
8674: 096, 048, 024, 012, 006, 000, 156
8680: 000, 240, 048, 048, 048, 048, 152
8686: 048, 240, 000, 008, 028, 054, 104
8692: 099, 000, 000, 000, 000, 000, 087
8698: 000, 000, 000, 000, 000, 255, 249
8704: 000, 000, 000, 000, 000, 000
8710: 000, 124, 194, 153, 153, 118
8716: 129, 153, 153, 230, 252, 130, 035
8722: 153, 130, 153, 153, 131, 252, 222
8728: 124, 194, 153, 158, 158, 153, 196
Entering SpeedScript

9034:024,000,056,024,024,060,006
9040:024,240,024,000,024,024,160
9046:024,024,000,000,192,204,018
9052:216,248,204,198,000,000,190
9058:056,024,024,024,024,060,054
9064:000,000,000,204,254,254,048
9070:214,198,000,000,000,252,006
9076:198,198,198,198,000,000,140
9082:000,124,198,198,198,124,196
9088:192,192,000,252,198,198,136
9094:198,252,006,066,000,126,210
9100:198,198,198,198,266,000,092
9106:000,252,198,192,192,192,148
9112:000,000,000,000,252,192,082
9118:006,252,000,000,048,254,206
9124:048,048,048,030,000,000,082
9130:000,198,198,198,198,000,064
9136:000,000,000,198,198,198,002
9142:108,056,000,000,000,198,032
9148:214,254,124,108,000,000,120
9160:006,252,000,198,198,198,028
9166:198,126,000,000,000,254,016
9172:012,056,096,254,014,000,132
9178:014,024,024,056,024,024,128
9184:024,024,024,024,024,024,148
9190:024,024,012,000,012,240,014
9196:024,028,024,024,024,000,080
9202:000,008,024,056,024,008,096
9208:000,000,000,016,016,024,048
9214:028,024,000,000,000,000,050
9220:000,000,000,000,000,000,004
9226:000,000,000,000,000,000,010
9232:165,128,141,048,036,165,187
9238:129,141,049,036,165,130,160
9244:141,051,036,165,131,141,181
9250:052,036,166,133,240,032,181
9256:169,000,141,115,063,160,176
9262:000,185,255,255,153,255,125
9268:255,200,204,115,063,208,073
9274:244,238,049,036,238,052,147
9280:036,224,000,240,007,202,005
9286:208,224,165,132,208,222,205
9292:096,165,133,170,005,132,009
9298:208,001,096,024,138,101,138
9304:129,141,120,036,165,128,039
9310:141,119,036,024,138,101,141
9316:131,141,123,036,165,130,058
9322:141,122,036,232,164,132,165
9328:208,004,240,013,160,255,224
9334: 185, 255, 255, 153, 255, 255, 196
9340: 136, 192, 255, 208, 245, 206, 086
9346: 120, 036, 206, 123, 036, 202, 085
9352: 208, 234, 096, 169, 040, 200, 059
9358: 024, 109, 108, 068, 024, 101, 064
9364: 088, 133, 136, 165, 089, 105, 096
9370: 000, 133, 137, 024, 173, 111, 220
9376: 063, 133, 138, 173, 112, 063, 074
9382: 133, 139, 162, 001, 173, 114, 120
9388: 063, 133, 145, 160, 000, 177, 082
9394: 138, 153, 123, 063, 200, 041, 128
9400: 127, 201, 094, 240, 022, 204, 048
9406: 107, 068, 208, 239, 136, 177, 101
9412: 138, 041, 127, 201, 000, 240, 175
9418: 007, 136, 208, 245, 172, 107, 053
9430: 000, 185, 123, 063, 145, 136, 098
9436: 200, 196, 140, 208, 246, 024, 210
9448: 139, 105, 000, 133, 139, 224, 204
9454: 001, 208, 003, 140, 110, 063, 251
9460: 204, 107, 068, 240, 008, 169, 016
9466: 064, 145, 136, 200, 076, 244, 091
9472: 036, 024, 165, 136, 105, 040, 250
9478: 133, 136, 144, 002, 230, 137, 020
9484: 232, 224, 019, 240, 003, 076, 038
9490: 175, 036, 165, 138, 141, 121, 026
9496: 063, 165, 139, 141, 122, 063, 205
9502: 096, 173, 102, 063, 133, 138, 223
9508: 141, 111, 063, 141, 117, 063, 160
9514: 133, 134, 173, 103, 063, 133, 013
9520: 139, 141, 112, 063, 141, 118, 250
9526: 063, 133, 135, 056, 173, 105, 207
9532: 063, 237, 103, 063, 170, 169, 097
9538: 000, 160, 255, 198, 139, 145, 195
9544: 138, 200, 230, 139, 145, 138, 038
9550: 200, 208, 251, 230, 139, 202, 028
9556: 208, 246, 145, 138, 096, 133, 026
9562: 140, 132, 141, 169, 001, 141, 046
9568: 240, 002, 160, 000, 177, 140, 047
9574: 240, 006, 032, 127, 047, 200, 242
9580: 208, 246, 096, 032, 204, 047, 173
9586: 240, 251, 096, 032, 064, 021, 050
9598: 000, 165, 144, 145, 134, 032, 234
9604: 234, 037, 076, 072, 038, 169, 246
9610: 125, 032, 127, 047, 169, 000, 126
9616: 141, 114, 063, 141, 102, 063, 000
9622: 141, 104, 063, 141, 106, 063, 000
9628: 141, 108, 063, 141, 245, 063, 149

SpeedScript
Entering SpeedScript

9634:141,020,064,141,182,067,009
9640:141,190,002,141,108,068,050
9646:169,040,141,107,068,169,100
9652:068,024,105,001,141,103,110
9658:063,173,049,002,056,233,250
9664:001,141,109,063,056,233,027
9670:008,141,107,063,056,233,038
9676:001,141,105,063,169,255,170
9682:141,243,063,165,075,240,113
9688:016,173,109,063,141,105,055
9694:063,169,007,141,107,063,004
9700:169,030,141,109,063,096,068
9706:032,173,045,173,102,063,054
9712:133,134,173,103,063,133,211
9718:135,032,139,036,032,010,118
9724:038,169,152,160,061,032,096
9730:089,037,238,113,063,076,106
9736:207,039,032,026,038,169,007
9742:136,160,061,032,089,037,017
9748:169,000,141,113,063,096,090
9754:160,039,169,000,145,088,115
9760:136,016,251,169,000,133,225
9766:082,133,088,133,084,096,139
9772:072,041,128,133,140,104,150
9778:041,127,201,096,176,013,192
9784:201,032,176,006,024,105,088
9790:064,076,069,038,056,233,086
9796:032,005,140,096,160,000,245
9802:140,106,068,177,134,133,064
9808:144,160,000,140,184,067,007
9814:177,134,073,128,145,134,109
9820:173,106,068,073,001,141,142
9826:106,068,032,139,036,032,255
9832:204,047,208,040,169,008,012
9838:141,031,208,173,031,208,134
9844:201,006,208,015,160,000,194
9850:140,106,068,165,144,145,122
9856:134,032,161,043,076,072,134
9862:038,165,020,041,016,240,142
9868:218,169,000,133,020,076,244
9874:081,038,170,169,008,141,241
9880:031,208,173,031,208,201,236
9886:005,208,005,169,128,141,065
9892:184,067,160,000,165,144,116
9898:145,134,173,113,063,240,014
9904:007,138,072,032,010,038,217
9910:104,170,138,201,155,208,134
9916:005,162,030,076,226,038,213
9922:138,044,182,067,048,026,187
9928:201,156,176,102,041,127,235
<table>
<thead>
<tr>
<th>Row</th>
<th>Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9934: 201, 032, 144, 096, 201, 123, 235</td>
</tr>
<tr>
<td>1</td>
<td>9940: 176, 092, 201, 092, 240, 088, 077</td>
</tr>
<tr>
<td>1</td>
<td>9946: 201, 094, 240, 084, 201, 095, 109</td>
</tr>
<tr>
<td>1</td>
<td>9952: 240, 080, 138, 072, 160, 000, 146</td>
</tr>
<tr>
<td>1</td>
<td>9958: 140, 182, 067, 177, 134, 201, 107</td>
</tr>
<tr>
<td>1</td>
<td>9964: 094, 240, 005, 173, 114, 063, 157</td>
</tr>
<tr>
<td>1</td>
<td>9970: 240, 003, 032, 124, 044, 104, 021</td>
</tr>
<tr>
<td>1</td>
<td>9976: 032, 044, 038, 041, 127, 013, 031</td>
</tr>
<tr>
<td>1</td>
<td>9982: 184, 067, 160, 000, 145, 134, 176</td>
</tr>
<tr>
<td>1</td>
<td>9988: 032, 139, 036, 056, 165, 134, 054</td>
</tr>
<tr>
<td>1</td>
<td>9994: 237, 117, 063, 133, 140, 165, 097</td>
</tr>
<tr>
<td>2</td>
<td>10000: 135, 237, 118, 063, 005, 140, 202</td>
</tr>
<tr>
<td>2</td>
<td>10006: 144, 014, 165, 134, 105, 000, 072</td>
</tr>
<tr>
<td>2</td>
<td>10012: 141, 117, 063, 165, 135, 105, 242</td>
</tr>
<tr>
<td>2</td>
<td>10018: 000, 141, 118, 063, 238, 134, 208</td>
</tr>
<tr>
<td>2</td>
<td>10024: 008, 002, 230, 135, 032, 287, 086</td>
</tr>
<tr>
<td>2</td>
<td>10030: 039, 076, 072, 038, 174, 038, 016</td>
</tr>
<tr>
<td>2</td>
<td>10036: 039, 221, 083, 039, 240, 006, 168</td>
</tr>
<tr>
<td>2</td>
<td>10042: 202, 208, 248, 076, 072, 038, 134</td>
</tr>
<tr>
<td>2</td>
<td>10048: 202, 138, 010, 170, 169, 038, 023</td>
</tr>
<tr>
<td>2</td>
<td>10054: 072, 169, 071, 072, 189, 120, 251</td>
</tr>
<tr>
<td>2</td>
<td>10060: 039, 072, 189, 119, 039, 072, 094</td>
</tr>
<tr>
<td>2</td>
<td>10066: 096, 035, 031, 030, 092, 094, 204</td>
</tr>
<tr>
<td>2</td>
<td>10072: 002, 020, 028, 029, 126, 255, 036</td>
</tr>
<tr>
<td>2</td>
<td>10078: 004, 009, 125, 124, 095, 005, 200</td>
</tr>
<tr>
<td>2</td>
<td>10084: 012, 019, 013, 018, 024, 026, 212</td>
</tr>
<tr>
<td>2</td>
<td>10090: 016, 254, 001, 011, 006, 021, 159</td>
</tr>
<tr>
<td>2</td>
<td>10096: 127, 157, 003, 007, 156, 027, 077</td>
</tr>
<tr>
<td>2</td>
<td>10102: 015, 132, 040, 183, 040, 236, 252</td>
</tr>
<tr>
<td>2</td>
<td>10108: 040, 034, 041, 130, 041, 138, 036</td>
</tr>
<tr>
<td>2</td>
<td>10114: 041, 154, 041, 000, 042, 049, 201</td>
</tr>
<tr>
<td>2</td>
<td>10120: 043, 123, 044, 091, 043, 225, 193</td>
</tr>
<tr>
<td>2</td>
<td>10126: 044, 001, 045, 048, 045, 081, 150</td>
</tr>
<tr>
<td>2</td>
<td>10132: 045, 050, 046, 056, 053, 092, 234</td>
</tr>
<tr>
<td>2</td>
<td>10138: 052, 186, 049, 124, 054, 016, 123</td>
</tr>
<tr>
<td>2</td>
<td>10144: 055, 102, 041, 189, 055, 076, 166</td>
</tr>
<tr>
<td>2</td>
<td>10150: 043, 032, 055, 079, 042, 132, 037</td>
</tr>
<tr>
<td>2</td>
<td>10156: 059, 109, 061, 083, 044, 075, 091</td>
</tr>
<tr>
<td>2</td>
<td>10162: 044, 047, 060, 099, 059, 216, 191</td>
</tr>
<tr>
<td>2</td>
<td>10168: 043, 188, 039, 197, 039, 173, 095</td>
</tr>
<tr>
<td>2</td>
<td>10174: 182, 067, 073, 128, 141, 182, 195</td>
</tr>
<tr>
<td>2</td>
<td>10180: 067, 096, 173, 004, 034, 073, 131</td>
</tr>
<tr>
<td>2</td>
<td>10186: 016, 141, 004, 034, 096, 032, 013</td>
</tr>
<tr>
<td>2</td>
<td>10192: 045, 040, 056, 165, 134, 237, 117</td>
</tr>
<tr>
<td>2</td>
<td>10198: 111, 063, 165, 135, 237, 112, 013</td>
</tr>
<tr>
<td>2</td>
<td>10204: 063, 176, 032, 056, 173, 111, 063</td>
</tr>
<tr>
<td>2</td>
<td>10210: 063, 237, 102, 063, 133, 140, 196</td>
</tr>
<tr>
<td>2</td>
<td>10216: 173, 112, 063, 237, 103, 063, 215</td>
</tr>
<tr>
<td>2</td>
<td>10222: 005, 140, 240, 013, 165, 134, 167</td>
</tr>
<tr>
<td>2</td>
<td>10228: 141, 111, 063, 165, 135, 141, 232</td>
</tr>
</tbody>
</table>
10234: 112, 063, 032, 139, 036, 056, 176
10240: 173, 121, 063, 229, 134, 133, 085
10246: 138, 173, 122, 063, 229, 135, 098
10252: 133, 139, 005, 138, 240, 002, 157
10258: 176, 024, 024, 173, 111, 063, 077
10264: 109, 110, 063, 141, 111, 063, 109
10270: 173, 112, 063, 105, 000, 141, 112
10276: 112, 063, 032, 139, 036, 076, 238
10282: 255, 039, 096, 056, 173, 117, 010
10288: 063, 237, 104, 063, 133, 140, 020
10294: 173, 118, 063, 237, 105, 063, 045
10300: 005, 140, 144, 012, 173, 104, 126
10306: 063, 141, 117, 063, 173, 105, 216
10312: 063, 141, 118, 063, 056, 165, 166
10318: 134, 237, 102, 063, 133, 140, 119
10324: 165, 135, 237, 103, 063, 005, 024
10330: 140, 176, 011, 173, 102, 063, 243
10336: 133, 134, 173, 103, 063, 133, 067
10342: 135, 096, 056, 165, 134, 237, 157
10348: 117, 063, 133, 140, 165, 135, 093
10354: 237, 118, 063, 005, 140, 176, 085
10360: 001, 096, 173, 117, 063, 133, 191
10366: 134, 173, 118, 063, 133, 135, 114
10372: 096, 169, 008, 141, 031, 208, 017
10378: 173, 031, 208, 201, 003, 208, 194
10384: 039, 173, 107, 068, 201, 040, 251
10390: 240, 020, 238, 107, 068, 238, 037
10396: 107, 068, 206, 108, 068, 032, 233
10402: 139, 036, 032, 207, 039, 169, 016
10408: 125, 032, 127, 047, 076, 010, 073
10414: 038, 230, 134, 208, 002, 230, 248
10420: 135, 076, 207, 039, 169, 008, 046
10426: 141, 031, 208, 173, 031, 208, 210
10432: 201, 003, 208, 030, 173, 107, 146
10438: 068, 201, 002, 240, 020, 206, 167
10444: 107, 068, 206, 107, 068, 238, 230
10450: 103, 068, 032, 139, 036, 032, 113
10456: 207, 039, 169, 125, 032, 127, 147
10462: 047, 076, 010, 038, 165, 134, 180
10468: 208, 002, 198, 135, 198, 134, 079
10474: 076, 207, 039, 165, 134, 133, 220
10480: 138, 165, 135, 133, 139, 198, 124
10486: 139, 160, 255, 177, 138, 201, 036
10492: 000, 240, 004, 201, 094, 208, 231
10498: 003, 136, 208, 243, 177, 138, 139
10504: 201, 000, 240, 008, 201, 094, 240
10510: 240, 004, 136, 208, 243, 096, 173
10516: 056, 152, 101, 138, 133, 134, 222
10522: 165, 139, 105, 000, 133, 135, 191
10528: 076, 207, 039, 160, 000, 177, 179
Entering SpeedScript

11434: 165, 134, 133, 128, 109, 238, 053
11440: 063, 133, 130, 165, 135, 133, 167
11446: 129, 109, 239, 063, 133, 131, 218
11452: 056, 173, 117, 063, 229, 128, 186
11458: 133, 132, 173, 118, 063, 229, 018
11464: 129, 133, 133, 032, 077, 036, 228
11470: 024, 173, 117, 063, 109, 238, 162
11476: 063, 141, 117, 063, 133, 131, 218
11482: 056, 173, 117, 063, 229, 128, 186
11488: 133, 132, 173, 118, 063, 229, 018
11494: 129, 133, 133, 032, 077, 036, 228
11500: 024, 173, 117, 063, 109, 238, 162
11506: 063, 141, 117, 063, 133, 131, 218
11512: 056, 173, 117, 063, 229, 128, 186
11518: 133, 132, 173, 118, 063, 229, 018
11524: 129, 133, 133, 032, 077, 036, 228
11530: 024, 173, 117, 063, 109, 238, 162
11536: 063, 141, 117, 063, 133, 131, 218
11542: 056, 173, 117, 063, 229, 128, 186
11548: 133, 132, 173, 118, 063, 229, 018
11554: 076, 010, 038, 162, 250, 154, 212
11560: 032, 031, 208, 173, 031, 208, 028
11566: 076, 072, 038, 160, 000, 177, 057
11572: 134, 201, 094, 240, 017, 200, 170
11578: 208, 247, 230, 135, 165, 135, 154
11584: 205, 118, 063, 144, 238, 240, 048
11590: 236, 076, 090, 041, 200, 208, 153
11596: 002, 230, 135, 076, 075, 041, 123
11602: 165, 134, 133, 138, 165, 135, 184
11608: 133, 139, 198, 139, 160, 255, 088
11614: 177, 138, 201, 094, 240, 017, 193
11620: 136, 192, 255, 208, 245, 198, 054
11626: 139, 165, 139, 205, 103, 063, 152
11632: 176, 236, 076, 244, 041, 056, 173
11644: 000, 101, 139, 133, 139, 056, 180
11650: 165, 138, 229, 134, 133, 140, 045
11656: 165, 139, 229, 135, 005, 140, 181
11662: 208, 018, 132, 140, 024, 165, 061
11668: 138, 229, 140, 133, 138, 165, 067
11674: 139, 233, 000, 133, 139, 076, 106
11680: 100, 045, 165, 138, 133, 134, 107
11686: 165, 139, 133, 135, 076, 207, 253
11692: 039, 169, 064, 141, 014, 212, 043
11698: 169, 010, 141, 000, 002, 169, 157
11704: 046, 141, 001, 002, 173, 048, 083
11710: 002, 133, 140, 173, 049, 002, 177
11716: 133, 141, 160, 000, 185, 238, 029
11722: 045, 145, 140, 200, 192, 028, 184
11728: 208, 246, 160, 004, 165, 088, 055

49
Entering *SpeedScript*

12034: 231, 046, 201, 155, 240, 069, 176
12040: 201, 126, 208, 015, 136, 016, 198
12046: 004, 200, 076, 231, 046, 169, 228
12052: 126, 032, 127, 047, 076, 231, 147
12058: 046, 133, 140, 041, 127, 201, 202
12064: 032, 144, 196, 201, 125, 176, 138
12070: 192, 204, 119, 063, 240, 187, 019
12076: 165, 140, 041, 127, 162, 008, 175
12082: 142, 031, 208, 174, 031, 208, 076
12088: 224, 005, 208, 002, 009, 128, 120
12094: 153, 163, 063, 032, 127, 047, 135
12100: 169, 000, 141, 182, 067, 200, 059
12106: 076, 231, 046, 162, 001, 142, 220
12112: 240, 002, 169, 000, 153, 163, 039
12118: 063, 152, 096, 162, 000, 169, 216
12124: 012, 141, 066, 003, 032, 006, 176
12130: 228, 162, 000, 169, 153, 141, 183
12136: 068, 003, 169, 047, 141, 069, 089
12142: 003, 169, 002, 141, 072, 003, 244
12148: 142, 073, 003, 169, 003, 157, 151
12154: 066, 003, 076, 086, 228, 140, 209
12160: 203, 047, 162, 000, 142, 072, 242
12166: 003, 142, 073, 003, 142, 255, 240
12172: 002, 160, 011, 140, 066, 003, 010
12178: 032, 086, 228, 172, 203, 047, 146
12184: 096, 069, 058, 160, 128, 076, 227
12190: 162, 047, 160, 000, 140, 104, 003
12196: 068, 134, 212, 133, 213, 032, 188
12202: 170, 217, 032, 230, 216, 160, 171
12208: 000, 177, 243, 072, 041, 127, 068
12214: 044, 104, 068, 048, 006, 032, 228
12220: 127, 047, 076, 196, 047, 032, 201
12226: 098, 055, 104, 048, 003, 200, 190
12232: 208, 231, 096, 018, 173, 252, 154
12238: 002, 201, 255, 208, 003, 169, 020
12244: 000, 096, 173, 252, 002, 201, 168
12250: 255, 240, 249, 141, 109, 068, 000
12256: 169, 255, 141, 252, 002, 133, 152
12262: 017, 032, 041, 048, 173, 109, 138
12268: 068, 201, 192, 176, 016, 041, 162
12274: 063, 201, 060, 208, 024, 173, 203
12280: 109, 068, 041, 064, 240, 006, 008
12286: 141, 190, 002, 169, 000, 096, 084
12292: 173, 190, 002, 073, 064, 141, 135
12298: 190, 002, 169, 000, 096, 174, 129
12304: 109, 068, 189, 064, 048, 044, 026
12310: 190, 002, 080, 010, 201, 097, 090
12316: 144, 006, 201, 123, 176, 002, 168
12322: 041, 223, 201, 128, 240, 217, 060
12328: 096, 072, 169, 050, 141, 000, 056
12334: 210, 162, 175, 142, 001, 210, 178
12340: 160, 128, 136, 208, 253, 202, 115
12346: 224, 159, 208, 243, 104, 096, 068
12352: 108, 106, 059, 128, 128, 107, 188
12358: 043, 042, 111, 128, 112, 117, 111
12364: 155, 105, 045, 061, 118, 128, 176
12370: 099, 128, 128, 098, 120, 122, 009
12376: 052, 128, 051, 054, 027, 053, 197
12382: 050, 049, 044, 032, 046, 110, 169
12388: 128, 109, 047, 128, 114, 128, 242
12394: 101, 121, 127, 116, 119, 113, 035
12400: 057, 128, 048, 055, 126, 056, 070
12406: 060, 062, 102, 104, 100, 128, 162
12412: 130, 103, 115, 097, 076, 074, 207
12418: 058, 128, 128, 075, 092, 094, 193
12424: 079, 128, 080, 085, 155, 073, 224
12430: 095, 124, 086, 128, 067, 128, 002
12436: 128, 066, 088, 090, 036, 128, 172
12442: 035, 038, 027, 037, 034, 033, 102
12448: 091, 032, 093, 078, 128, 077, 147
12454: 063, 128, 082, 128, 069, 089, 213
12460: 159, 084, 087, 081, 040, 128, 239
12466: 041, 039, 156, 064, 125, 157, 248
12472: 070, 072, 068, 128, 131, 071, 212
12478: 083, 065, 012, 010, 123, 128, 099
12484: 128, 011, 030, 031, 015, 128, 027
12490: 016, 021, 155, 009, 028, 029, 204
12496: 022, 128, 003, 128, 128, 002, 107
12502: 024, 026, 128, 128, 133, 128, 013
12508: 027, 128, 253, 128, 000, 032, 020
12514: 096, 014, 128, 013, 128, 128, 221
12520: 018, 128, 005, 025, 158, 020, 074
12526: 023, 017, 128, 128, 128, 128, 022
12532: 254, 128, 125, 255, 006, 008, 252
12538: 004, 128, 132, 007, 019, 001, 029
12544: 032, 132, 049, 162, 112, 169, 144
12550: 122, 157, 068, 003, 169, 062, 075
12556: 157, 069, 003, 169, 005, 157, 060
12562: 072, 003, 169, 000, 157, 073, 236
12568: 003, 169, 006, 157, 074, 003, 180
12574: 169, 003, 157, 066, 003, 032, 204
12580: 086, 228, 048, 092, 169, 004, 147
12586: 141, 103, 068, 174, 103, 068, 187
12592: 165, 100, 157, 229, 067, 165, 163
12598: 101, 157, 230, 067, 238, 103, 182
12604: 068, 238, 103, 068, 032, 169, 226
12610: 049, 048, 063, 201, 043, 176, 134
12616: 075, 032, 127, 047, 032, 169, 042
12622: 049, 048, 051, 169, 000, 141, 024

52
Entering SpeedScript

12628: 105, 068, 032, 169, 049, 048, 043
12634: 041, 032, 127, 047, 238, 105, 168
12640: 068, 173, 105, 068, 201, 008, 207
12646: 208, 008, 169, 046, 032, 127, 180
12652: 047, 076, 086, 049, 201, 011, 066
12658: 208, 226, 169, 005, 133, 140, 227
12664: 032, 169, 049, 198, 140, 165, 105
12670: 140, 208, 247, 076, 045, 049, 123
12676: 162, 112, 169, 012, 157, 066, 042
12682: 003, 032, 086, 228, 162, 112, 249
12688: 188, 067, 003, 096, 072, 169, 227
12694: 155, 032, 127, 047, 104, 032, 135
12700: 127, 047, 032, 169, 049, 048, 116
12706: 225, 032, 127, 047, 076, 158, 059
12712: 049, 162, 112, 169, 000, 157, 049
12718: 072, 003, 157, 073, 003, 169, 139
12724: 007, 157, 066, 003, 076, 086, 063
12730: 228, 032, 255, 053, 032, 089, 107
12736: 047, 032, 255, 053, 169, 001, 237
12742: 141, 240, 002, 133, 082, 169, 197
12748: 125, 032, 127, 047, 032, 000, 055
12754: 049, 032, 239, 051, 173, 229, 215
12760: 067, 133, 136, 173, 230, 067, 254
12766: 133, 137, 169, 000, 141, 228, 006
12772: 067, 206, 103, 068, 206, 103, 213
12778: 068, 032, 227, 051, 032, 111, 243
12784: 037, 162, 001, 142, 240, 002, 056
12790: 174, 025, 050, 201, 097, 144, 169
12796: 002, 041, 095, 133, 140, 221, 116
12802: 025, 050, 240, 006, 202, 208, 221
12808: 248, 076, 242, 050, 202, 138, 196
12814: 010, 170, 189, 042, 050, 072, 035
12820: 189, 041, 050, 072, 096, 015, 227
12826: 030, 031, 028, 029, 004, 082, 230
12832: 076, 085, 070, 049, 050, 051, 157
12838: 052, 027, 012, 070, 050, 083, 076
12844: 050, 112, 050, 129, 050, 228, 151
12850: 050, 007, 051, 247, 050, 255, 198
12856: 050, 084, 051, 110, 051, 110, 000
12862: 051, 110, 051, 110, 051, 152, 075
12868: 051, 118, 051, 032, 227, 051, 086
12874: 174, 228, 067, 240, 031, 202, 248
12880: 202, 076, 097, 050, 032, 227, 252
12886: 051, 174, 228, 067, 232, 232, 046
12892: 236, 103, 068, 176, 013, 142, 062
12898: 228, 067, 189, 229, 067, 133, 243
12904: 136, 189, 230, 067, 133, 137, 228
12910: 076, 235, 049, 032, 227, 051, 012
12916: 173, 228, 067, 201, 006, 144, 167
12922: 243, 056, 233, 006, 170, 076, 138
Entering SpeedScript

13222:038,076,072,038,169,022,069
13228:133,084,169,157,032,127,106
13234:047,076,127,047,140,236,083
13240:063,032,132,049,032,170,150
13246:051,169,050,160,062,032,202
13252:089,037,174,236,063,169,196
13258:000,032,160,047,169,253,095
13270:062,032,089,037,032,111,065
13276:037,032,239,051,076,242,129
13282:050,160,012,177,136,073,066
13288:128,145,136,136,016,247,016
13294:096,032,170,051,169,000,244
13300:160,063,032,089,037,173,030
13306:123,062,076,127,047,169,086
13312:000,141,183,067,076,012,223
13318:052,169,128,141,183,067,234
13324:173,102,063,133,138,173,026
13330:103,063,133,139,076,038,058
13336:052,169,000,141,183,067,124
13342:165,134,133,138,165,135,132
13348:133,139,056,173,118,063,206
13354:229,139,170,232,160,000,204
13360:177,138,044,183,067,048,193
13366:015,201,155,208,005,169,039
13372:094,076,082,052,032,044,184
13378:038,076,082,052,201,094,097
13384:208,005,169,155,076,082,255
13390:052,032,056,055,145,138,044
13396:200,208,217,230,139,202,000
13402:208,212,096,032,026,038,190
13408:169,026,160,062,032,089,122
13414:037,169,008,032,235,052,123
13420:048,064,032,007,052,162,217
13426:112,173,102,063,157,068,021
13432:003,173,103,063,157,069,176
13438:003,056,173,117,063,237,007
13444:102,063,157,072,003,173,190
13450:118,063,237,103,063,157,111
13456:073,003,169,011,157,066,111
13462:003,032,086,228,048,011,046
13468:032,255,051,032,132,049,195
13474:048,010,076,232,053,152,221
13480:072,032,255,051,104,168,082
13486:192,128,240,033,152,072,223
13492:169,125,032,127,047,169,081
13498:050,160,062,032,089,037,104
13504:104,170,169,000,032,160,059
13510:047,032,224,052,032,173,246
13516:045,169,001,141,113,063,224
13522: 096, 032, 026, 038, 169, 058, 117
13528: 160, 062, 032, 089, 037, 076, 160
13534: 199, 052, 174, 185, 067, 169, 044
13540: 012, 157, 066, 003, 076, 086, 116
13546: 228, 162, 112, 142, 185, 067, 106
13552: 141, 186, 067, 173, 190, 002, 231
13558: 072, 169, 064, 141, 190, 002, 116
13564: 032, 206, 046, 104, 141, 190, 203
13570: 002, 173, 120, 063, 208, 008, 064
13576: 032, 010, 038, 104, 104, 076, 116
13582: 173, 045, 032, 224, 052, 174, 202
13588: 185, 067, 169, 163, 157, 068, 061
13594: 003, 169, 063, 157, 069, 003, 234
13600: 173, 120, 063, 157, 072, 003, 108
13606: 169, 000, 157, 073, 003, 173, 101
13612: 186, 067, 157, 074, 003, 169, 188
13618: 003, 157, 066, 003, 076, 086, 185
13624: 228, 056, 165, 134, 237, 102, 210
13630: 063, 133, 138, 133, 142, 165, 068
13636: 135, 237, 103, 063, 133, 139, 110
13642: 133, 143, 005, 138, 240, 004, 225
13648: 169, 196, 133, 145, 032, 026, 013
13654: 038, 169, 084, 160, 062, 032, 119
13660: 089, 037, 169, 004, 032, 235, 146
13666: 052, 016, 003, 076, 174, 052, 215
13672: 165, 145, 201, 196, 240, 003, 030
13678: 032, 031, 037, 032, 126, 053, 165
13684: 192, 128, 144, 003, 076, 174, 065
13690: 052, 076, 232, 053, 174, 185, 126
13696: 067, 165, 134, 157, 068, 003, 210
13702: 165, 135, 157, 069, 003, 056, 207
13708: 173, 104, 063, 229, 134, 157, 232
13714: 072, 003, 173, 105, 063, 229, 023
13720: 135, 157, 073, 003, 169, 007, 184
13726: 157, 066, 003, 032, 086, 228, 218
13732: 016, 005, 192, 136, 240, 001, 242
13738: 096, 174, 185, 067, 024, 189, 137
13744: 072, 003, 109, 102, 063, 141, 154
13750: 117, 063, 189, 073, 003, 109, 224
13756: 103, 063, 141, 118, 063, 024, 188
13762: 173, 117, 063, 101, 142, 141, 163
13768: 117, 063, 173, 118, 063, 101, 067
13774: 143, 141, 118, 063, 032, 025, 216
13780: 052, 173, 117, 063, 133, 138, 120
13786: 173, 118, 063, 133, 139, 169, 245
13792: 000, 168, 145, 138, 200, 208, 059
13798: 251, 096, 032, 224, 052, 016, 133
13804: 003, 076, 174, 052, 169, 125, 067
13810: 032, 127, 047, 169, 074, 160, 083
13816: 062, 032, 089, 037, 076, 199, 231
Entering *SpeedScript*

```
13822:052,169,064,141,014,212,138
13828:173,138,041,141,198,002,185
13834:141,200,002,173,154,041,209
13840:141,197,002,096,162,000,102
13846:142,205,063,142,206,063,075
13852:142,207,063,142,208,063,085
13858:056,177,138,233,016,144,030
13864:042,201,010,176,038,014,009
13870:205,063,046,206,063,014,131
13876:205,063,046,206,063,014,137
13882:205,063,046,206,063,014,143
13888:205,063,046,206,063,013,148
13894:205,063,141,205,063,200,179
13900:208,212,230,139,076,034,207
13906:054,248,173,205,063,013,070
13912:206,063,240,028,056,173,086
13918:205,063,233,001,141,205,174
13924:063,173,206,063,233,000,070
13930:141,206,063,238,207,063,000
13936:208,003,238,208,063,076,140
13942:084,054,173,207,063,216,147
13948:096,056,173,209,063,237,190
13954:106,063,141,211,063,173,119
13960:210,063,237,107,063,141,189
13966:212,063,013,211,063,208,144
13972:016,032,026,038,169,140,057
13978:160,062,032,089,037,169,191
13984:001,141,113,063,096,024,086
13990:165,134,133,128,109,211,022
13996:063,133,130,165,135,133,163
14002:129,109,212,063,133,131,187
14008:056,173,117,063,229,128,182
14014:133,132,173,118,063,229,014
14020:129,133,133,024,101,131,079
14026:205,105,063,144,016,032,255
14032:026,038,169,128,160,062,023
14038:032,089,037,169,001,141,171
14044:113,063,096,032,077,036,125
14050:024,173,211,063,133,132,194
14056:109,117,063,141,117,063,074
14062:173,212,063,133,133,109,037
14068:118,063,141,118,063,165,144
14074:134,133,130,165,135,133,056
14080:131,173,106,063,133,128,222
14086:173,107,063,133,129,032,131
14092:016,036,076,207,039,160,034
14098:000,177,134,170,200,177,108
14104:134,136,145,134,200,138,143
14110:145,134,006,160,000,177,230
14116:134,041,063,201,033,144,140
```
<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
<th>Column 6</th>
<th>Column 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>14716:063,240,006,032,098,055,106</td>
<td>59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
14722: 136, 208, 250, 096, 172, 226, 194
14728: 063, 024, 152, 109, 235, 063, 014
14734: 141, 235, 063, 032, 152, 057, 054
14740: 136, 208, 250, 096, 169, 155, 138
14746: 032, 098, 055, 173, 181, 067, 248
14752: 240, 003, 032, 098, 055, 096, 172
14758: 141, 237, 063, 041, 127, 032, 039
14764: 056, 055, 174, 241, 057, 221, 208
14770: 241, 057, 240, 009, 202, 208, 111
14776: 248, 206, 234, 063, 076, 246, 233
14782: 058, 202, 138, 010, 170, 140, 140
14788: 236, 063, 169, 057, 072, 169, 194
14794: 212, 072, 189, 004, 058, 072, 041
14800: 189, 003, 058, 072, 096, 056, 170
14806: 173, 236, 063, 101, 138, 133, 034
14812: 138, 165, 139, 105, 000, 133, 132
14818: 139, 076, 020, 056, 177, 138, 064
14824: 201, 094, 240, 001, 136, 140, 020
14830: 236, 063, 096, 017, 119, 108, 109
14836: 114, 116, 098, 115, 110, 104, 133
14842: 102, 064, 112, 063, 120, 109, 052
14848: 105, 103, 106, 097, 058, 115, 172
14854: 058, 125, 058, 135, 058, 145, 073
14860: 058, 155, 058, 165, 058, 180, 174
14866: 058, 214, 058, 071, 158, 087, 052
14872: 058, 055, 058, 045, 058, 036, 078
14878: 058, 239, 058, 024, 059, 106, 062
14884: 058, 200, 169, 000, 141, 233, 195
14890: 063, 076, 230, 057, 200, 032, 188
14896: 020, 054, 141, 232, 063, 076, 122
14902: 230, 057, 200, 032, 020, 054, 135
14908: 141, 230, 063, 173, 208, 063, 170
14914: 141, 231, 063, 076, 230, 057, 096
14920: 200, 032, 020, 054, 141, 228, 235
14926: 063, 173, 208, 063, 141, 229, 187
14932: 063, 076, 230, 057, 200, 032, 230
14938: 020, 054, 141, 223, 063, 076, 155
14944: 230, 057, 169, 000, 141, 227, 152
14950: 063, 200, 076, 230, 057, 169, 129
14956: 010, 141, 181, 067, 200, 076, 015
14962: 230, 057, 200, 032, 020, 054, 195
14968: 141, 221, 063, 076, 230, 057, 140
14974: 200, 032, 020, 054, 141, 222, 027
14980: 063, 076, 230, 057, 200, 032, 022
14986: 020, 054, 141, 224, 063, 076, 204
14992: 230, 057, 200, 032, 020, 054, 225
14998: 141, 225, 063, 076, 230, 057, 174
15004: 200, 032, 020, 054, 141, 226, 061
15010: 063, 076, 230, 057, 172, 236, 228
15016: 063, 200, 152, 072, 032, 238, 157
<table>
<thead>
<tr>
<th>SpeedScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>15316: 138, 240, 005, 224, 000, 208, 003</td>
</tr>
<tr>
<td>15322: 235, 202, 200, 208, 011, 230, 024</td>
</tr>
<tr>
<td>15328: 139, 165, 139, 205, 118, 063, 029</td>
</tr>
<tr>
<td>15334: 240, 002, 176, 054, 232, 236, 146</td>
</tr>
<tr>
<td>15340: 245, 063, 208, 221, 024, 152, 125</td>
</tr>
<tr>
<td>15346: 101, 138, 133, 140, 165, 139, 034</td>
</tr>
<tr>
<td>15352: 105, 000, 133, 141, 173, 117, 149</td>
</tr>
<tr>
<td>15358: 063, 197, 140, 173, 118, 063, 240</td>
</tr>
<tr>
<td>15364: 229, 141, 144, 024, 056, 165, 251</td>
</tr>
<tr>
<td>15370: 140, 237, 245, 063, 133, 134, 194</td>
</tr>
<tr>
<td>15376: 141, 242, 063, 165, 141, 233, 233</td>
</tr>
<tr>
<td>15382: 000, 133, 135, 141, 243, 063, 225</td>
</tr>
<tr>
<td>15388: 032, 207, 039, 096, 032, 026, 204</td>
</tr>
<tr>
<td>15394: 038, 169, 235, 160, 062, 032, 218</td>
</tr>
<tr>
<td>15400: 089, 037, 169, 001, 141, 113, 078</td>
</tr>
<tr>
<td>15406: 063, 096, 169, 008, 141, 031, 042</td>
</tr>
<tr>
<td>15412: 208, 173, 031, 208, 201, 003, 108</td>
</tr>
<tr>
<td>15418: 208, 035, 032, 026, 038, 169, 054</td>
</tr>
<tr>
<td>15424: 245, 160, 062, 032, 089, 037, 177</td>
</tr>
<tr>
<td>15430: 032, 206, 046, 141, 020, 064, 067</td>
</tr>
<tr>
<td>15436: 240, 014, 160, 000, 185, 163, 070</td>
</tr>
<tr>
<td>15442: 063, 153, 021, 064, 206, 204, 019</td>
</tr>
<tr>
<td>15448: 120, 063, 208, 244, 076, 010, 041</td>
</tr>
<tr>
<td>15454: 038, 056, 165, 134, 133, 130, 238</td>
</tr>
<tr>
<td>15460: 237, 242, 063, 133, 140, 165, 056</td>
</tr>
<tr>
<td>15466: 135, 133, 131, 237, 243, 063, 024</td>
</tr>
<tr>
<td>15472: 005, 140, 208, 101, 169, 255, 222</td>
</tr>
<tr>
<td>15478: 141, 243, 063, 024, 173, 245, 239</td>
</tr>
<tr>
<td>15484: 063, 101, 134, 133, 128, 169, 084</td>
</tr>
<tr>
<td>15490: 000, 101, 135, 133, 129, 056, 172</td>
</tr>
<tr>
<td>15496: 173, 117, 063, 229, 130, 133, 213</td>
</tr>
<tr>
<td>15502: 132, 173, 118, 063, 229, 131, 220</td>
</tr>
<tr>
<td>15508: 133, 133, 032, 016, 036, 056, 042</td>
</tr>
<tr>
<td>15514: 173, 117, 063, 237, 245, 063, 028</td>
</tr>
<tr>
<td>15520: 141, 117, 063, 173, 118, 063, 067</td>
</tr>
<tr>
<td>15526: 233, 000, 141, 118, 063, 173, 126</td>
</tr>
<tr>
<td>15532: 020, 064, 240, 041, 141, 238, 148</td>
</tr>
<tr>
<td>15538: 063, 169, 000, 141, 239, 063, 085</td>
</tr>
<tr>
<td>15544: 032, 146, 044, 239, 185, 239</td>
</tr>
<tr>
<td>15550: 021, 064, 032, 044, 038, 145, 022</td>
</tr>
<tr>
<td>15556: 134, 200, 204, 020, 064, 208, 002</td>
</tr>
<tr>
<td>15562: 242, 024, 165, 134, 109, 020, 128</td>
</tr>
<tr>
<td>15568: 064, 133, 134, 165, 135, 105, 176</td>
</tr>
<tr>
<td>15574: 000, 133, 135, 076, 207, 039, 036</td>
</tr>
<tr>
<td>15580: 160, 000, 204, 115, 063, 240, 234</td>
</tr>
<tr>
<td>15586: 029, 177, 142, 048, 026, 032, 168</td>
</tr>
<tr>
<td>15592: 056, 055, 032, 098, 055, 173, 189</td>
</tr>
<tr>
<td>15598: 241, 063, 240, 016, 169, 008, 201</td>
</tr>
<tr>
<td>15604: 032, 098, 055, 169, 095, 032, 213</td>
</tr>
<tr>
<td>15610: 098, 055, 200, 076, 222, 060, 193</td>
</tr>
</tbody>
</table>
Entering SpeedScript

15616:096,140,236,063,041,127,191
15622:141,237,063,032,056,055,078
15628:201,099,208,027,056,173,008
15634:232,063,237,115,063,074,034
15640:056,237,221,063,168,169,170
15646:032,032,098,055,136,208,079
15652:250,172,236,063,076,252,061
15658:060,201,101,208,017,056,173
15664:173,222,063,237,115,063,153
15670:056,237,221,063,168,169,200
15676:032,076,031,061,201,117,066
15682:208,008,173,241,063,073,064
15688:001,141,241,063,201,035,242
15694:208,018,140,236,063,174,149
15700:228,063,173,229,063,032,104
15706:155,047,172,236,063,076,071
15712:252,060,174,237,063,189,047
15718:051,064,032,098,055,076,222
15724:252,060,032,026,038,056,060
15730:173,104,063,237,117,063,103
15736:170,173,105,063,237,118,218
15742:063,032,160,047,169,001,086
15748:141,113,063,096,083,112,228
15754:101,101,100,083,099,114,224
15760:105,112,116,032,051,046,094
15766:048,000,032,098,121,032,225
15778:115,032,066,114,097,110,184
15784:110,111,110,000,066,117,170
15790:102,102,101,114,032,067,180
15796:108,101,097,114,101,100,033
15802:000,066,117,102,102,101,162
15820:101,032,040,083,044,087,079
15826:044,080,041,000,058,032,209
15832:065,114,101,032,121,111,248
15838:117,032,115,117,114,101,050
15844:063,032,040,089,047,078,065
15850:041,058,000,069,082,065,037
15856:083,069,032,065,076,076,129
15862:032,084,069,088,084,000,091
15868:069,114,097,115,101,032,012
15874:040,083,044,087,044,080,124
15880:041,058,032,210,197,212,246
15886:213,210,206,032,116,111,134
15892:032,101,120,105,116,000,238
15898:083,097,118,101,032,040,241
15904:068,101,118,105,099,101,112
15910:058,070,105,108,101,110,078
15916:097,109,101,041,062,000,198
15922:069,114,114,111,114,032,092
15928:035,000,066,082,069,065,117
15934:075,032,075,101,121,032,242
15940:065,098,111,114,116,000,060
15946:078,111,032,069,114,114,080
15952:111,114,115,000,076,111,095
15958:097,100,032,040,068,101,012
15964:118,105,099,101,058,070,131
15970:105,108,101,110,097,109,216
15976:101,041,062,000,032,080,164
15982:114,101,115,115,032,210,029
15988:197,212,213,210,206,000,130
15994:068,049,058,042,046,042,171
16000:077,101,109,111,114,121,249
16006:032,070,117,108,108,000,057
16012:078,111,032,116,101,120,186
16018:116,032,105,110,032,098,127
16024:117,102,102,101,114,000,176
16030:080,114,105,110,116,032,203
16036:040,068,101,118,105,099,183
16048:110,097,109,101,041,062,184
16054:000,080,114,105,110,116,195
16060:105,110,103,046,046,046,132
16066:155,155,000,073,110,115,034
16072:101,114,116,032,110,101,006
16078:120,116,032,115,104,101,026
16084:101,116,044,032,112,114,219
16090:101,115,115,032,210,197,220
16096:212,213,210,206,000,070,111
16102:105,110,100,058,000,078,169
16108:111,116,032,102,111,117,057
16114:110,100,000,067,104,097,208
16120:110,103,101,032,116,111,053
16126:058,000,027,028,027,029,167
16132:027,030,027,031,032,195,090
16138:212,210,204,045,196,101,210
16144:108,101,116,101,032,204,166
16150:111,099,107,032,213,110,182
16156:108,111,099,107,032,210,183
16162:101,110,097,109,101,032,072
16168:197,211,195,198,111,114,042
16174:109,097,116,032,195,212,039
16180:210,204,045,204,111,097,155
16186:100,032,032,068,114,105,253
16192:118,101,032,091,177,032,103
16198:178,032,179,032,180,093,252
Entering SpeedScript

16204:058,032,000,082,101,110,203
16210:097,109,101,032,116,111,136
16216:058,000,070,111,114,109,038
16222:097,116,032,100,105,115,147
16228:107,000,000,000,000,000,000,207

16.234
3936
3.98
Chapter 3

SpeedScript
Source Code
Atari Source Code

The source code for SpeedScript was originally developed using the MAC/65 assembler (from Optimized Systems Software, Inc.). The MAC/65 assembler uses the standard MOS source code format, so this source code can be assembled on a variety of Atari assemblers, including EASMD from OSS and the Atari Assembler/Editor cartridge. The source code was originally broken up into a number of modules, each SAVE#’d to disk. The .INCLUDE pseudo-op was used to link all the modules together. All files must be merged together to be assembled with the Atari Assembler/Editor cartridge. Line numbers are omitted.

Most pseudo-ops are in standard MOS 6502 notation: *= updates the program counter (some assemblers use .ORG instead); .BYTE assembles a list of numbers or an ATASCII character string; .WOR, or .WORD, assembles a list of addresses into low byte/high byte format; < extracts the low byte of a 16-bit expression; > extracts the high byte of a 16-bit expression (some assemblers reverse the use of < and >; others, such as EASMD and the Assembler/Editor cartridge, use a suffix of &255 and /256 to achieve the same effect); and = is used to assign an expression to a label (some assemblers use .EQU).

Beginners should make sure they understand Indirect-Y addressing, as in LDA ($FB),Y or LDA (CURR),Y. This mode is used extensively in SpeedScript.

The Atari version of SpeedScript was developed by sending the Commodore 64 source code to the Atari via modem. References to Commodore 64 Kernal ROM routines were replaced with Atari CIO routines. Some routines built into the Commodore 64’s ROM had to be programmed into Atari SpeedScript, with resulting code expansion. References to location 1 (which maps banks of ROM in and out in the 64) were omitted. The REFRESH routine, TOPCLR, and a few other routines were changed to compensate for Atari’s floating screen memory. The raster interrupt used to highlight the command line in the 64 version became a display-list interrupt. A custom character set was added to take advantage of the Atari’s special nine-line character mode. The DOS package was written to support disk functions. But much of the source code did not need to be changed at all, since SpeedScript’s
machine-specific code is segregated into distinct modules. These modules were rewritten. Approximately one week was required to get a primitive version running, followed by two months of testing, debugging, and refining to complete Atari SpeedScript. Because of the new character set, the DOS package, smoother input/output programming (such as Atari's device-independent I/O), and more logical keyboard layout, the Atari version may be the best version of SpeedScript yet.

SpeedScript is written in small modules. Some people think that subroutines are useful only when a routine is called more than once. I strongly believe in breaking up a problem into a number of discrete tasks. These tasks can be written as subroutines, then tested individually. Once all the modules are working, just link them together with JSRs and you have a working program.

I've also tried to use meaningful labels, but sometimes one just runs out of imagination. Comments are added below as signposts to guide you through the source code (you needn't type them in—if you do, precede each comment with a semicolon for the sake of your assembler). Modules are also set apart with blank lines. Notice that some modules are used in rather creative ways. For example, word left/word right is used both for moving the cursor and in delimiting a word to be erased in the erase mode. Also, note that memory locations are sometimes used instead of meaningful labels. In order to fit the complete source code into memory at once, I sometimes had to compromise readability for the sake of brevity.

Crucial to the understanding of SpeedScript is the REFRESH routine. Study it carefully. REFRESH is the only routine in SpeedScript that writes directly to the screen (CIO is used to print on the command line). It automatically takes care of word-wrap and carriage returns, and provides useful pointers so that the CHECK routine can easily scroll the screen. This frees the rest of SpeedScript to just move and modify contiguous memory. Carriage returns are not padded out in memory with spaces to fill the rest of a line; the REFRESH routine takes care of this transparently.
SpeedScript 3.0 Source Code for Atari

Filename: D:SPEED.O

Location $1F00 is safely above DOS 2.05, DOS 3, and OS/A+ DOS. Some DOS's may use more memory, so you may need to reassemble SpeedScript at a higher address, usually the address of LOMEM plus 256 bytes to be safe.

\* = $1F00

Locations used by high-speed memory move routines.

FROMML = $80
FROMMH = $81
DESTL = $82
DESTH = $83
LLEN = $84
HLEN = $85

CURR: Position of cursor within text memory. SCR: used by the REFRESH routine.

CURR = $86
SCR = $88

TEX: An alternate location used in tandem with CURR. COLR is used by REFRESH. TEMP is used throughout as a scratchpad pointer. INDIR is also a reusable indirect pointer. UNDERCURS stores the value of the character highlighted by the cursor.

TEX = $8A
TEMP = $8C
INDIR = $8E
UNDERCURS = $90

WINDCOLR: Color of command line window supported by HIGHLIGHT. RETCHAR is the screen-code value of the return-mark (a left-pointing arrow). SPACE is the screen-code value of the space character. RED and BLUE are used as command-line colors

WINDCOLR = $91
RETCCHAR = 94
SPACE = 0
RED = $32
BLUE = $74

We save the DOS reset vector and change this vector to point to SpeedScript's SYSTEM RESET routine. Since this routine is called at power-up, right after DOS.SYS runs, we need to disable the cold-start flag (location 580) and set location $09 to signify a successful disk boot.

LDA 710
STA $709
JSR INIT
LDA #$8C
CMP FIRSTRUN
STA FIRSTRUN
BEQ SKIPERAS
JSR ERASE
JSR KILLBUFF

The character set for the ANTIC 3 nine-line character mode must be on an even 512-byte boundary, so we force the assembler's program counter to address $2000 and merge in the character set. We then link in each successive module of SpeedScript. Again, if your assembler cannot handle .INCLUDE,
you'll have to merge all these files together in the order indicated.

* = $2000

```
INCLUDE #D:CHSET.SRC
INCLUDE #D:SPEED.1
INCLUDE #D:SUPPORT
INCLUDE #D:DOSPAK
INCLUDE #D:SPEED.2
INCLUDE #D:DATA
END
```

Filename D:CHSET.SRC

The character set here is stored as eight bytes per line, so each line defines one character. Sheldon Leemon's INSTEDIT character editor was used to create the character set, and I wrote a special program to convert the character set into .BYTE statements. In ANTIC mode 3, each character takes up ten scan lines of vertical screen space. The characters in the lowercase portion of the character set are displayed with a blank line at the top line, then the character data from bytes 1–7 of the character set. Byte 0 of the character's definition is displayed at the ninth line of the character. The tenth line is always blank. This lets you define characters with true descenders. The forced blank line lets you use more of the character matrix for defining a character, so these characters are larger than normal Atari characters.

```
.BYTE 0,0,0,0,0,0,0
.BYTE 0,0,0,0,0,0,0,0
.BYTE 0,0,0,0,0,0,0,0
.BYTE 0,0,0,0,0,0,0,0
.BYTE 0,0,0,0,0,0,0,0
.BYTE 0,0,0,0,0,0,0,0
.BYTE 0,0,0,0,0,0,0,0
```


UMOVE is a high-speed memory move routine. It gets its speed from self-modifying code (the $FFFFs at
MOVLOOP are replaced by actual addresses when UMOVE is called).
UMOVE is used to move an overlapping range of memory upward (toward location 0), so it is used to delete. Set
FROML/FROMH to point to the source area of memory, DESTL/DESTH to point to the destination, and
LLEN/HLEN to hold the length of the area being moved.

UMOVE
LDA FROML
STA MOVLOOP+1
LDA FROMH
STA MOVLOOP+2
LDA DESTL
STA MOVLOOP+4
LDA DESTH
STA MOVLOOP+5
LDX HLEN
BEQ SKIPMOV

MOV1
LDA #0
MOV2
STA ENDPOS
LDY #0

MOVLOOP
LDA $FFFF,Y
STA $FFFF,Y
INY
CPY ENDPOS
BNE MOVLOOP
INC MOVLOOP+2
INC MOVLOOP+5
CPX #0
BEQ OUT
DEX
BNE MOV1

SKIPMOV
LDA LLEN
BNE MOV2

OUT
RTS

DMOVE uses the same variables as UMOVE, but it is used to move an
overlapping block of memory downward (toward location $FFFF), so it is
used to insert. If the block of memory
to be moved does not overlap the
destination area, then either routine can
be used.

DMOVE
LDA HLEN
TAX
ORA LLEN
BNE NOTNULL
RTS

NOTNULL
CLC
TXA
ADC FROMH

Filename D:SPEED.1

This module is chiefly concerned with
the word processor editing functions. It
contains many common subroutines,
such as TOPCLR and PRMSG to clear
the command line and print messages.
It contains the initialization routines
and takes care of memory moves (in-
serts and deletes). A second module,
SPEED.2, is responsible for most
input/output, including the printer
routines. SPEED.1 is the largest file in
the linked chain.
REFRESH copies a screenful of text from the area of memory pointed to by TOPLIN. It works like a printer routine, fitting a line of text between the screen margins, wrapping words, and restarting at the left margin after printing a carriage return. SpeedScript constantly calls this routine while the cursor is blinking. To eliminate flicker, it clears out the end of each line instead of first clearing the screen. It stores the length of the first screen line for the sake of the CHECK routine (which scrolls up by adding that length to TOPLIN) and the last text location referenced (so CHECK can see if the cursor has moved off the visible screen). REFRESH can automatically handle different screen widths.

REFRESH
LDA #40
INY
CLC

RLM: Left margin. Location $58/$59 points to the address of screen memory.

ADC RLM
CLC
ADC $58
STA SCR
LDA $59
ADC #0
STA SCR+1

Character #64 (ATASCII value of 0) fills the gap when a line is broken. It can be redefined to show or not show these false spaces.

LDA #64
STA (SCR),Y
INY
JMP CLRNL
CLC

TOPLIN points to the first character within text to be printed at the top-left corner of the screen.

$FFFF,Y
$FFFF,Y
#255
DMOVLOOP
DMOV1
DMOVLOOP
SKIPDMOV

SLOOP
NXCUR
BREAK
COPY

LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y
LDA (TEX),Y

INY
INY
INY
INY

BREAK
COPY

LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y
LDA LBUFF,Y

CLC
CLC
CLC
CLC

ADC TEX
ADC TEX
ADC TEX
ADC TEX
ADC TEX
ADC TEX
ADC TEX
ADC TEX
ADC TEX
ADC TEX
ADC TEX
ADC TEX
ADC TEX
ADC TEX
ADC TEX
ADC TEX

STA TEX
STA TEX
STA TEX
STA TEX
STA TEX
STA TEX
STA TEX
STA TEX
STA TEX
STA TEX
STA TEX
STA TEX
STA TEX
STA TEX
STA TEX
STA TEX

STM: Left margin. Location $58/$59 points to the address of screen memory.

ADC LRM
CLC
ADC $58
STA SCR
LDA $59
ADC #0
STA SCR+1

CLEAR
The following routine fills the entire text area with space characters (screen code 0), effectively erasing all text. It is called when the program is first run and when an Erase All is performed. It also initializes the cursor position (CURR) and the end-of-text pointer (LASTLINE).

```
ERASE
LDA TEXSTART
STA TEX
STA TOPLIN
STA LASTLINE
STA CURR
LDA TEXSTART+1
STA TEX+1
STA TOPLIN+1
STA LASTLINE+1
STA CURR+1
SEC
LDA TEXEND+1
SBC TEXSTART+1
TAX
LDA #SPACE
LDY #255
DEC TEX+1
STA (TEX),Y
INY
INC TEX+1
STA (TEX),Y
INY
BNE CLR2
INC TEX+1
DEX
BNE CLR2
STA (TEX),Y
RTS
```

PRMSG is used anytime we need to print something at the top of the screen (the command line). Pass it the address of the message to be printed by storing the low byte of the address in the accumulator and the high byte in the Y register. The message in memory must end with a zero byte. The routine does not add a carriage return. CHROUT is a subroutine in the SUP-PORT package.

```
PRMSG
STA TEMP
STY TEMP+1
LDA #1
STA 752
LDY #0
PRLOOP
LDA (TEMP),Y
BEQ PREXIT
JSR CHROUT
INY
BNE RTX
PREXIT
RTS
GETAKEY
JSR GETIN
BEQ GETKEY
RTS
JDOS
JSR PREXIT
LDA BLINK
BEQ NOBLINK
LDY #0
LDA UNDERCURS
STA (CURR),Y
NOBLINK
JSR INIT
JMP MAIN
INIT
LDA #125
JSR CHROUT
LDA #0
STA INSMODE
STA TEXSTART
STA TEXEND
STA TEXBUF
STA BUEND
STA HUNTLN
STA REPLEN
STA ESCFLAG
STA SHFLOK
STA RLM
LDA #40
STA LINELEN
```

Label END is at the end of the source code, so it points to the last address used by the object code. We use it to calculate the start-of-text memory.

```
LDA # >END
CLC
ADC #1
STA TEXSTART+1
```

Location 561 points to the display list, which holds screen information at the top of memory. We use it as the last address available for storing text or buffer text.
If location $4B$ is 0, then *SpeedScript* is booted from disk. If we booted from cassette, we free up the DOS area ($0700$-$0E00$) for use as the text buffer, and free up the text memory used by disk-based *SpeedScript* as the text buffer.

The second initialization routine turns on the display-list interrupt (HIGHLIGHT), homes the cursor, and prints the credit line.

 SYSMSG displays “SpeedScript 3.0.” The message flag (MSGFLG) is set when a message is to be left on the screen only until the next keystroke. After that keystroke, SYSMSG is called. The INIT2 routine prints the credit line with the MSGFLG set so that you won’t have to stare at the author’s name while you’re writing—a modesty feature.

The MAIN loop blinks the cursor, checks for keystrokes, converts them from ATASCII to screen codes, puts them in text at the CURRrent position, and increments the CURRrent position and LASTLINE. It also checks for special cases like the RETURN key and passes control characters to the CON¬TROL routine. The INSMODE flag is checked to see if we should insert a space before a character.

The MAIN loop blinks the cursor, checks for keystrokes, converts them from ATASCII to screen codes, puts them in text at the CURRrent position, and increments the CURRrent position and LASTLINE. It also checks for special cases like the RETURN key and passes control characters to the CONTROL routine. The INSMODE flag is checked to see if we should insert a space before a character.
We check for the START key, and if pressed, go to the HOME cursor routine.

The realtime clock (location 20), which counts in 1/60 seconds, is checked for 16/60 seconds (about 1/5 second) to see if it's time to blink the cursor.

A key has been pressed. We check the SELECT key to see if the keystroke should be inverted.

Put the character into memory.

Move the cursor forward.

CONTROL looks up a keyboard command in the list of control codes at CTBL. The first byte of CTBL is the
actual number of commands. Once the position is found, this position is doubled as an index to the two-byte address table at VECT. The address of MAIN-1 is put on the stack, simulating the return address; then the address of the command routine taken from VECT is pushed. We then perform an RTS. RTS pulls the bytes off the stack as if they were put there by a JSR. This powerful technique is used to simulate ON-GOTO in machine language.

<table>
<thead>
<tr>
<th>CONTROL</th>
<th>LDX</th>
<th>CTBL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRCH</td>
<td>CMP</td>
<td>CTBL,X</td>
</tr>
<tr>
<td>BEQ</td>
<td>FOUND</td>
<td></td>
</tr>
<tr>
<td>DEX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE</td>
<td>SRC</td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>MAIN</td>
<td></td>
</tr>
</tbody>
</table>

FOUND

| DEX      |     |       |
| TXA      | ASL | A     |
| TAX      |     |       |
| LDA      | # >MAIN-1|
| PHA      |     |       |
| LDA      | # <MAIN-1|
| PHA      |     |       |
| LDA      | VECT+1,X|
| PHA      |     |       |
| LDA      | VECT,X|
| PHA      |     |       |
| RTS      |     |       |

CTBL

 BYTE 35
 BYTE 31,30,92,94,2,20,28,29
 BYTE 126,255,4
 BYTE 9,125,124,95,5,12,19
 BYTE 13,18,24,26,16
 BYTE 254,1,11,6,21,127,157
 BYTE 3,7,156,27,15

VECT .WORD RIGHT-1,LEFT-1, WLEFT-1, WRIGHT-1, BORDER-1, LET TER,S-1
 .WORD SLEFT-1, SRIGHT-1, DELCHAR-1, INSC, HAR-1, DELETE-1
 .WORD INSTGL-1, CLEAR-1, PARIGHT-1, PA RLEFT-1
 .WORD ERAS-1, ILOAD-1, TSAVE-1
 .WORD DOS-1, INSBUFFER-1, SWITCH-1
 .WORD ENDTEX-1, PRINT-1
 .WORD DELIN-1, ALPHA-1, KILLBUFFER-1, HUN T-1, FREEMEM-1, T AB-1

.WORD LOTTSPACE-1, RE
 PSTART-1, SANDR
 -1, EATSSPACE-1, E
 SC-1, OONOFF-1

Toggle ESCape mode.

ESC

 LDA ESCFLAG
 EOR #128
 STA ESCFLAG
 RTS

Change the character definition of the character used to fill in the end of a line. It alternates between being a blank space, and being a blank space with a tiny dot visible. This lets you see which spaces are actually part of your text and which are just used to parse the screen. Beware of the address $2204 if you reassemble at a different address (sorry, I didn’t use a label).

ONOFF

 LDA $2204
 EOR #16
 STA $2204
 RTS

The CHECK routine first prevents the cursor from disappearing past the beginning or end-of-text memory and prevents us from cursoring past the end-of-text pointer. It also checks to see if the cursor has left the visible screen, scrolling with REFRESH to make the cursor visible. The double-byte SBCs are used as a 16-bit CMP macro, setting the Z and C flags just like CMP does.

CHECK

 JSR CHECK2
 SEC
 LDA CURR
 SBC TOPLIN
 LDA CURR+1
 SBC TOPLIN+1
 BCS OK1
 SEC
 LDA TOPLIN
 SBC TEXSTART
 STA TEMP
 SBC TEXSTART+1
 ORA TEMP
 BEQ OK1
 LDA CURR
 STA TOPLIN
 LDA CURR+1
 STA TOPLIN+1
 JSR REFRESH
 SEC
 LDA BOTSCR
 SBC CURR

OK1
STA TEX
LDA BOTSCR+1
SBC CURR+1
STA TEX+1
ORA TEX
BEQ EQA
BCS OK2
CLC
LDA TOPLIN
ADC LENTABLE
STA TOPLIN
LDA TOPLIN+1
ADC #0
STA TOPLIN+1
JSR REFRESH
STA TOPLIN
LDA TOPLIN+1
ORA TEX
BEQ EQA
BEQ NOBIGGER
BCS OK2
INC LINELEN
:,
EQA CLC INC LINELEN
LDA TOPLIN
OEC
AOC LENTABLE
JSR REFRESH
STA TOPLIN
JSR CHECK
LDA #125
JSR CHROUT
STA TOPLIN
JSR CHECK
STA TOPLIN+1
LDA #125
JSR CHROUT
STA TOPLIN+1
NOBIGGER
JMP SYMSMG
CRIGHT
INC CURR
BNE NOINCR
INC CURR+1
NOINCR
JMP CHECK

Move cursor left. If the OPTION key is held down, we instead decrease the line length.

LEFT
LDA #8
STA 53279
LDA 53279
CMP #3
BNE CRIGHT
LDA LENELEN
CMP #48
BEQ NOBIGGER
INC LENELEN
INC LENELEN
DEC RLM
JSR REFRESH
JSR CHECK
LDA #125
JSR CHROUT
JSR REFRESH
STA TOPLIN
JSR CHECK
LDA #125
JSR CHROUT
STA TOPLIN
LDA #125
JSR CHROUT
STA TOPLIN
NOOEC
STA CURR
LDA CURR+1
STA CURR+1
NOOEC
STA CURR+1
STA CURR+1
INRANGE
SEC
LDA CURR
SBC TEXSTART
STA TEMP
LDA CURR+1
SBC TEXSTART+1
ORA TEMP
BCS INRANGE
LDA TEXSTART
STA CURR
LDA TEXSTART+1
STA CURR+1
RTS

Word left. We look backward for a space.

WLEFT
LDA CURR
STA TEX
LDA CURR+1
STA TEX+1
DEC TEX+1
LDY #$FF

Move cursor right. If the OPTION key is held down, we instead increase the line length.

RIGHT
LDA #8
STA 53279

STRIP
LDA (TEX),Y
CMP #SPACE
BEQ STRLOOP
CMP #RETCCHAR
BNE WLOOP
DEY
BNE STRIP
WLOOP
LDA (TEX),Y
CMP #SPACE
BEQ WROUT

STRLOOP

Word right. We scan forward for a space. OIDS is not a meaningful label.

**WRIGHT**

RLOOP

LDY #0
LDA (CURR),Y
CMP #SPACE
BEQ ROUT
CMP #RETCHAR
BEQ ROUT
INY
BNE RLOOP
RTS
INY
BNE OIDS
INC CURR+1
LDA CURR+1
CMP LASTLINE+1
BCC OIDS
BNE LASTWORD
LDA (CURR),Y
CMP #SPACE
BEQ ROUT
CMP #RETCHAR
BEQ ROUT

Add the Y register to the CURRent cursor position to move the cursor. CHECK prevents illegal cursor movement. LASTWORD is called if the end of the word cannot be found before we reach the end-of-text.

**ADYCURRE**

CLC
TYA
ADC CURR
STA CURR
LDA CURR+1
ADC #0
STA CURR+1

**WRTN**

JMP CHECK

**LASTWORD**

LDA LASTLINE
STA CURR
LDA LASTLINE+1
STA CURR+1
JMP CHECK

**ENDTEX** is tricky. If the end-of-text pointer would point to an area already visible on the screen, we just move the cursor there and call REFRESH. Otherwise, we step back 1K from the end-of-text and then scroll to the end. This is necessary since in the worst case only 18 characters of return-marks would fill the screen.

**ENDTEX**

LDA #0
STA TOPLIN
LDA LASTLINE+1
SEC
SBC #4
CMP TEXTSTART+1
BCS SAFE
LDA TEXTSTART+1
SAFE
STA TOPLIN+1
JSR REFRESH
JMP LASTWORD

Change the border color. The display-list interrupt automatically places SCRCOL into the hardware background color register #2.

**BORDER**

INC SCRCOL
INC SCRCOL
RTS

**SCRCOL** .BYTE 8

Change text luminance. TEXCOLR is stored into hardware color register #1 during the display-list interrupt.

**LETTERS**

INC TEXCOLR
INC TEXCOLR
LDA TEXCOLR
AND #15
STA TEXCOLR
RTS

**TEXCOLR** .BYTE 2

Sentence left. We look backward for ending punctuation or a return-mark, then go forward until we run out of spaces.

**SLEFT**

LDA CURR
STA TEX
LDA CURR+1
STA TEX+1
DEC TEX+1
LDY #$FF

**PMANY**

LDA (TEX),Y
CMP #'.−32
BEQ PSRCH
CMP #1'−32
BEQ PSRCH
CMP #7'−32
BEQ PSRCH

**PSRCH**

BNE PSLOOP
DEY
BNE PMANY
### SpeedScript Source Code

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTS</td>
<td>LDA (TEX),Y</td>
<td>Initialize text buffer</td>
</tr>
<tr>
<td>PSLOOP</td>
<td>CMP #'.-32</td>
<td>Check for punctuation</td>
</tr>
<tr>
<td></td>
<td>BEQ PUNCT2</td>
<td>If punctuation, store in PUNCT2</td>
</tr>
<tr>
<td></td>
<td>CMP #'.-32</td>
<td>Check for punctuation</td>
</tr>
<tr>
<td></td>
<td>BEQ PUNCT2</td>
<td>If punctuation, store in PUNCT2</td>
</tr>
<tr>
<td></td>
<td>CMP #?-.32</td>
<td>Check for punctuation</td>
</tr>
<tr>
<td></td>
<td>BEQ PUNCT2</td>
<td>If punctuation, store in PUNCT2</td>
</tr>
<tr>
<td></td>
<td>CMP #RETCHAR</td>
<td>Check for return character</td>
</tr>
<tr>
<td></td>
<td>BEQ PUNCT2</td>
<td>If return character, store in PUNCT2</td>
</tr>
<tr>
<td></td>
<td>DEY</td>
<td>Decrement TEMP</td>
</tr>
<tr>
<td></td>
<td>BNE PSLOOP</td>
<td>If not empty, loop</td>
</tr>
<tr>
<td></td>
<td>CMP TEXSTART</td>
<td>Check if at start of text</td>
</tr>
<tr>
<td>PUNCT</td>
<td>LDA (TEX),Y</td>
<td>Store text in PUNCT2 buffer</td>
</tr>
<tr>
<td></td>
<td>CMP #SPACE</td>
<td>If space, continue</td>
</tr>
<tr>
<td></td>
<td>CMP #'.-32 BEQ PUNCT2</td>
<td>If punctuation, store in PUNCT2</td>
</tr>
<tr>
<td></td>
<td>CMP #'1-32 BEQ PUNCT2</td>
<td>If punctuation, store in PUNCT2</td>
</tr>
<tr>
<td></td>
<td>CMP #'7-32 BEQ PUNCT2</td>
<td>If punctuation, store in PUNCT2</td>
</tr>
<tr>
<td></td>
<td>CMP #RETCHAR BEQ PUNCT2</td>
<td>If return character, store in PUNCT2</td>
</tr>
<tr>
<td></td>
<td>DEY BEQ PUNCT2</td>
<td>If not empty, loop</td>
</tr>
<tr>
<td></td>
<td>BNE PSLOOP</td>
<td>If not empty, loop</td>
</tr>
<tr>
<td></td>
<td>IMP WROUT</td>
<td>Update cursor pointer</td>
</tr>
<tr>
<td></td>
<td>LDA #&lt;KILLMSG</td>
<td>Load kill message</td>
</tr>
<tr>
<td></td>
<td>IMP PSLOOP</td>
<td>Loop to display message</td>
</tr>
<tr>
<td></td>
<td>IMP PRMSG</td>
<td>Display message</td>
</tr>
<tr>
<td></td>
<td>LDA TEXSTART</td>
<td>Store start of text</td>
</tr>
<tr>
<td></td>
<td>LDA #1</td>
<td>Store flag</td>
</tr>
<tr>
<td></td>
<td>STA CURR+1</td>
<td>Set CURR to new start of text</td>
</tr>
</tbody>
</table>

The text buffer starts at a fixed location, but the end of the buffer is changed as text is added to it. To clear the buffer, we just set the end of the buffer to the value of the start of the buffer. No text is actually erased.

### Documentation Text

Sentence right. We look forward for ending punctuation, then skip forward until we run out of spaces.

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KILLBUFF</td>
<td>LDA TEXBUF</td>
<td>Load text buffer pointer</td>
</tr>
<tr>
<td></td>
<td>STA TPTR</td>
<td>Store text pointer</td>
</tr>
<tr>
<td></td>
<td>LDA TEXBUF+1</td>
<td>Load new text buffer pointer</td>
</tr>
<tr>
<td></td>
<td>STA TPTR+1</td>
<td>Store new text pointer</td>
</tr>
<tr>
<td></td>
<td>JSR TOPCLR</td>
<td>Call top-level delete routine</td>
</tr>
<tr>
<td></td>
<td>LDA #&lt;KILLMSG</td>
<td>Load kill message</td>
</tr>
<tr>
<td></td>
<td>LDY #&gt;KILLMSG</td>
<td>Load end of kill message</td>
</tr>
<tr>
<td></td>
<td>JSR PRMSG</td>
<td>Display message</td>
</tr>
<tr>
<td></td>
<td>STA MSGFLG</td>
<td>Store flag</td>
</tr>
<tr>
<td></td>
<td>RTS</td>
<td>Return to top-level delete routine</td>
</tr>
</tbody>
</table>

This is the second level of the general-purpose delete routines. UMOVE is the primitive core of deleting. For CTRL-D, the CURRent cursor position is the source; then a cursor command is called to update the cursor pointer. This becomes the destination. For CTRL-E, the CURRent cursor position is the destination; a cursor movement routine is called, and this becomes the source. UMOVE is then called. We actually move more than the length from the source to the end-of-text. Some extra text is moved from past the end-of-text. Since everything past the end-of-text is spaces, this neatly erases everything past the new end-of-text position.

Naturally, the end-of-text pointer is updated. Before the actual delete is performed, the text to be deleted is stored in the buffer so that it can be recalled in case of error. The buffer doubles as a fail-safe device, and for moving and copying text. Checks are made to make sure that the buffer does not overflow.
Most delete commands end up calling the above routines. The single-character deletes must subtract 1 from the buffer pointer so that single characters are not added to the buffer. But note how short these routines are.

Delete character (BACK S)

CTRL-BACK S

Called by CTRL-D. As mentioned, it stores CURR into FROML/FROMH, moves the cursor either by sentence, word, or paragraph, then stores the new position of CURR into DESTL and DESTH. The above routines perform the actual delete. CTRL-D always discards the previous contents of the buffer, for deleting text backward creates a buffer of out-of-order text. Notice how we change the color of the command window to red to warn the user of the impending deletion.
Home the cursor. This is called by the START key. We check to see if START is held down for at least 1/2 second. If it is, we move the cursor to the top of text.

**Home**

```
SEC
LDA CURR
SBC TOPLIN
STA TEMP
LDA CURR+1
SBC TOPLIN+1
ORA TEMP
BEQ TOPHOME
LDA TOPLIN
STA CURR
LDA TOPLIN+1
STA CURR+1
```

**WaitST**

```
LDA #0
STA 20
STA 53279
```

**HomePauses**

```
LDA 20
CMP #30
BNE HOMEPAUSE
```

**Outlet**

```
JMP CHECK
```

**Top**

```
LDA TXTSTART
STA CURR
LDA TXTSTART+1
STA CURR+1
JMP WAITST
```

This deletes all spaces between the cursor and following non-space text.

Sometimes inventing labels can be fun.

**Insert 255 spaces. Notice how it and other insert routines use TAB2.**

**Lottaspace**

```
LDA #255
STA INSLEN
JMP TAB2
```

**Tab**

```
LDA #5
STA INSLEN
JSR TAB2
LDA (CURR),Y
```
Toggle insert mode. The INSMODE flag doubles as the color of the command line.

Another example of modular code. This is called anytime a yes/no response is called for. It prints "Are you sure? (Y/N)," then returns with the zero flag set to true if Y was pressed, ready for the calling routine to use BEQ or BNE as a branch for yes or no. We trap out the clear-screen key in case this routine is called by Erase All, since otherwise repeating keys may instantly cancel the command. The AND #223 zaps out the distinction between uppercase and lowercase Y.

Erase all text. Allowed only if the OPTION key is held down with SHIFT-CLEAR. It calls YORN to affirm the deadly deed, then calls ERASE to erase all text, INIT2 to reset some flags, then jumps back to the MAIN loop. LDX #$FA / TXS is used to clean up the stack.

Insert a single space.

A general routine to insert as many spaces as are specified by INSLEN.
Paragraph right.

PARRIGHT
LDY #0
LDA (CURR),Y
CMP #RETCCHAR
BEQ RETFOUND
INY
BNE PARLP
INC CURR+1
LDA CURR+1
CMP LASTLINE+1
BCC PARLP
BEQ PARLP
JMP LASTWORD

RETFOUND
INY
BNE GOADY
INC CURR+1
JMP ADYCURR

Paragraph left. Notice the trick of decrementing the high byte of the pointer, then starting the index at 255 in order to search backward.

PARLEFT
LDA CURR
STA TEX
LDA CURR+1
STA TEX+1
DEC TEX+1
LDY #$FF

PARLOOP
LDA (TEX),Y
CMP #RETCCHAR
BEQ RETF2

PARCONT
DEY
CPY #255
BNE PARLOOP
DEC TEX+1
LDA TEX+1
CMP TEXSTART+1
BCS PARLOOP
JMP FIRSTWORD

RETF2
SEC
TYA
ADC TEX
STA TEX
LDA #0
ADC TEX+1
STA TEX+1
SEC
LDA TEX
SBC CURR
STA TEMP
LDA TEX+1
SBC CURR+1
ORA TEMP
BNE TEXTOCURR
STY TEMP
CLC

This enables the display-list interrupt (DLI). The DLI allows separate background colors for the command line and the rest of the screen. It lets us change the color of the top line to flag insert mode or to warn the user with a red color that he/she should be careful. Since it is an interrupt, it is always running in the background. Interrupt routines must always be careful not to corrupt the main program.

HIGHLIGHT turns off any DLIs (by storing #64 into $D40E), sets the NMI pointer ($200/$201), creates a custom display list of IRG mode 3 (lowercase descenders, GRAPHICS 0½) with DLI set in one line, then enables DLIs ($C0 into $D40E) and returns. The routine DLI is now running constantly in the background, changing the screen color of all text below the DLI.
The custom display list:

```
 .BYTE 3,3,3,3,3,16,65,0,0
```

The display-list interrupt routine stores the SCREEN COLOR and TEXT COLOR into the appropriate hardware registers, then stores the WINDOW COLOR into 710, and #10 into 709 to set the color of the top line of the screen. This line is automatically set by the normal vertical-blank interrupt. We also force the character-set pointer to keep our character set in place whenever we're on the editing screen.

```
PHA
LDA SCRCOL
STA #$D40A
STA #$D018
STA 712
LDA TEXCOLR
STA #$D017
LDA WINDCOLR
STA 710
LDA #10
STA 709
LDA #$20
STA 756
LDA #0
STA $02B6
PLA
RTI
```

ERAS is called by CTRL-E. It works much like CTRL-D. Notice that the ORA #64 allows users to press either S, W, P, or CTRL-S, CTRL-W, CTRL-P, in case they have a habit of leaving the control key held down. It must call REFRESH after each move and adjust the new position of the cursor. If OPTION is held down with CTRL-E, we don't erase the previous contents of the buffer, letting the user chain non-contiguous sections into the buffer for later recall.

```
ERAS
LDA #8
STA 33279
LDA 33279
CMP #3
BEQ ERAS1
JSR KILLBUFF
```

The INPUT routine is used to get responses from the command line. It returns the complete line in INBUFF. INLEN is the length of the input. A
Zero byte is stored at INBUFF+INLEN after the user presses RETURN. This routine is foolproof (I know...), since no control keys other than BACK S are allowed, unless preceded by ESCape. The SELECT key can be held down to enter inverse-video characters. The system cursor is turned on for this routine (by putting #0 into 752), then turned off when we exit (by putting #1 into 752). This routine also prevents the user from typing past the end of the command line. If the limit of typing length must be set arbitrarily, LIMIT is preset and INPUT is called at INP1. CURSIN is the MAIN loop.

```
INPUT   LDA #39
        SBC 85
        STA LIMIT

INP1    LDY #0
        STY INLEN
        STY 752
        LDA #32
        JSR CHROUT
        LDA #126
        JSR CHROUT

CURSIN  STY INLEN
        JSR GETAKEY
        LDY INLEN
        BIT ESCFLAG
        BMI ESCKEY
        CMP #27
        BNE NOESC
        LDA #128
        STA ESCFLAG
        STA $02A2
        JMP CURSIN

NOESC   CMP #155
        BEQ INEXIT
        CMP #126
        BNE NOBACK
        DEY
        BPL NOTZERO
        INY
        JMP CURSIN

NOTZERO LDA #126
        JSR CHROUT
        JMP CURSIN

NOBACK  STA TEMP
        AND #127
        CMP #32
        BCC CURSIN
        CMP #125
        BCS CURSIN
        CPY LIMIT
        BEQ CURSIN
        LDA TEMP

ESCKEY  AND #127
        LDX #8
        STX 53279
```

This module supports most primitive input/output functions, including a routine to clear the screen and reset the screen editor (OPENEDITOR), print a character (CHROUT), and get a key from the keyboard (GETAKEY).

```
OPENEDITOR LDX #0
        LDA #12
        STA ICCOM
        JSR CIO
        LDX #0
        LDA #<ENAME
        STA ICBADR
        LDA #>ENAME
        STA ICBADR+1
        LDA #2
        STA ICBLLEN
        STA ICBLLEN+1
        LDA #3
        STA ICCOM,X
        JMP CIO
```

```
Put the ATASCII value of the character into the accumulator and call CHROUT to print a character. The Y register is preserved. We call CIO with a buffer length of zero.

```
CHROUT STY CHRSAVE
        LDX #0
        STX ICBLLEN
        STX ICBLLEN+1
        STX $02FF
        LDY #11
        STY ICCOM
        JSR CIO
        LDY CHRSAVE
        RTS
```

The filename of the Editor device.

```
FILENAME D:SUPPORT.
```

```
This module supports most primitive input/output functions, including a routine to clear the screen and reset the screen editor (OPENEDITOR), print a character (CHROUT), and get a key from the keyboard (GETAKEY).
```

```
OPENEDITOR LDX #0
        LDA #12
        STA ICCOM
        JSR CIO
        LDX #0
        LDA #<ENAME
        STA ICBADR
        LDA #>ENAME
        STA ICBADR+1
        LDA #2
        STA ICBLLEN
        STA ICBLLEN+1
        LDA #3
        STA ICCOM,X
        JMP CIO
```

The filename of the Editor device.

```
FILENAME D:SUPPORT.
```

```
This module supports most primitive input/output functions, including a routine to clear the screen and reset the screen editor (OPENEDITOR), print a character (CHROUT), and get a key from the keyboard (GETAKEY).
```

```
OPENEDITOR LDX #0
        LDA #12
        STA ICCOM
        JSR CIO
        LDX #0
        LDA #<ENAME
        STA ICBADR
        LDA #>ENAME
        STA ICBADR+1
        LDA #2
        STA ICBLLEN
        STA ICBLLEN+1
        LDA #3
        STA ICCOM,X
        JMP CIO
```

The filename of the Editor device.

```
FILENAME D:SUPPORT.
```
OUTNUM and PROUTNUM print decimal numbers to the display or printer. The integer to be printed is passed with the low byte in the X register and the high byte in the accumulator. The integer to be printed is passed with the low byte in the X register and the high byte in the accumulator. The integer to be printed is passed with the low byte in the X register and the high byte in the accumulator. The integer to be printed is passed with the low byte in the X register and the high byte in the accumulator. The integer to be printed is passed with the low byte in the X register and the high byte in the accumulator. The integer to be printed is passed with the low byte in the X register and the high byte in the accumulator. The integer to be printed is passed with the low byte in the X register and the high byte in the accumulator. The integer to be printed is passed with the low byte in the X register and the high byte in the accumulator. The integer to be printed is passed with the low byte in the X register and the high byte in the accumulator. The integer to be printed is passed with the low byte in the X register and the high byte in the accumulator. The integer to be printed is passed with the low byte in the X register and the high byte in the accumulator.

The system keyboard fetch routine interferes with the display-list interrupt, since the blip of each key is timed with WSYNC, which freezes the ANTIC chip for one line. This causes annoying flicker. This routine uses POKEY sound decaying from volume 15 to 0 for the keyboard feedback tone. It’s not hard to create any sound effect you want for the keyboard blip. This routine mimics the system routine fairly closely. It’s easy to expand it to allow many more keyboard functions and full processing of new keystrokes just by changing some of this code and the keyboard table.

```
GETIN
LDA 764
CMP #$FF
BNE GETCHAR
LDA #0
RTS
GETCHAR
LDA 764
CMP #$FF
```

2F0D F

```
2FE2
BEQ GETCHAR
STA KEYVAL
LDA #$46D
STA 764
```

Clear break flag.

```
2FEC
STAt $11
JSR BLIP
LDA KEYVAL
```

Check for SHIFT+CTRL.

```
2FE7
CMP #$C0
BSC GIXIT
AND #63
CMP #$60
BNE NOTCAPS
LDA KEYVAL
AND #64
BEQ NOTSET
STA SHFLOK
```

The CAPS/LOWERR key toggles the SHiFTLOcK flag to allow either only uppercase, or both uppercase and lowercase.

```
2FE8
NOTSET
LDA SHFLOK
EOR #64
STA SHFLOK
LDA #0
RTS
```

```
NOTCAPS
LDA KEYBOARD,X
BIT SHFLOK
BVC NOTLOCKED
CMP #a
BCC NOTLOCKED
CMP #$2+1
BCS NOTLOCKED
AND #$223
NOTLOCKED
CMP #$80
BEQ GIXIT
RTS
```

The sound effect for the keyboard “blip.”

```
2FE9
BLIP
PHA
LDA #50
STA $D200
LDX #$AF
```

```
SNDLOOP
STX $D201
LDY #128
SLOW
DEY
BNE SLOW
DEX
CPX #$9F
BNE SNDLOOP
PLA
RTS
```

```
GETIN
LDA 764
CMP #$FF
BNE GETCHAR
LDA #0
RTS
GETCHAR
LDA 764
CMP #$FF
```

2F0D F

```
2FE2
BEQ GETCHAR
STA KEYVAL
LDA #$46D
STA 764
```

Clear break flag.

```
2FEC
STAt $11
JSR BLIP
LDA KEYVAL
```

Check for SHIFT+CTRL.

```
2FE7
CMP #$C0
BSC GIXIT
AND #63
CMP #$60
BNE NOTCAPS
LDA KEYVAL
AND #64
BEQ NOTSET
STA SHFLOK
```

The CAPS/LOWERR key toggles the SHiFTLOcK flag to allow either only uppercase, or both uppercase and lowercase.

```
2FE8
NOTSET
LDA SHFLOK
EOR #64
STA SHFLOK
LDA #0
RTS
```

```
NOTCAPS
LDA KEYBOARD,X
BIT SHFLOK
BVC NOTLOCKED
CMP #a
BCC NOTLOCKED
CMP #$2+1
BCS NOTLOCKED
AND #$223
NOTLOCKED
CMP #$80
BEQ GIXIT
RTS
```

The sound effect for the keyboard “blip.”

```
2FE9
BLIP
PHA
LDA #50
STA $D200
LDX #$AF
```

```
SNDLOOP
STX $D201
LDY #128
SLOW
DEY
BNE SLOW
DEX
CPX #$9F
BNE SNDLOOP
PLA
RTS
```

```
GETIN
LDA 764
CMP #$FF
BNE GETCHAR
LDA #0
RTS
GETCHAR
LDA 764
CMP #$FF
```

2F0D F

```
2FE2
BEQ GETCHAR
STA KEYVAL
LDA #$46D
STA 764
```

Clear break flag.

```
2FEC
STAt $11
JSR BLIP
LDA KEYVAL
```

Check for SHIFT+CTRL.

```
2FE7
CMP #$C0
BSC GIXIT
AND #63
CMP #$60
BNE NOTCAPS
LDA KEYVAL
AND #64
BEQ NOTSET
STA SHFLOK
```

The CAPS/LOWERR key toggles the SHiFTLOcK flag to allow either only uppercase, or both uppercase and lowercase.

```
2FE8
NOTSET
LDA SHFLOK
EOR #64
STA SHFLOK
LDA #0
RTS
```

```
NOTCAPS
LDA KEYBOARD,X
BIT SHFLOK
BVC NOTLOCKED
CMP #a
BCC NOTLOCKED
CMP #$2+1
BCS NOTLOCKED
AND #$223
NOTLOCKED
CMP #$80
BEQ GIXIT
RTS
```

The sound effect for the keyboard “blip.”

```
2FE9
BLIP
PHA
LDA #50
STA $D200
LDX #$AF
```

```
SNDLOOP
STX $D201
LDY #128
SLOW
DEY
BNE SLOW
DEX
CPX #$9F
BNE SNDLOOP
PLA
RTS
```
Filename D:DOSPAK

DOSPAK is a self-contained substitute for the DOS menu, although it uses several routines built into SpeedScript. The concept of DOSPAK is that all directory entries should fit on one screen. A large cursor is used to move from filename to filename. At any time, you can delete, rename, lock, unlock, or load the selected filename, just by pressing one key, or a CTRL key combination. Except for Rename, you don’t have to type the filename. You can also format the entire disk or redisplay the directory.

CATALOG fits the entire disk directory onto the screen by skipping over the sector counts, trimming up spacing, and placing three items per line. The cursor position of each filename is saved into a slot in memory so that the cursor routine can quickly and easily skip about.

```
.CATALOG
JSR CLOSE7
LDX #$70
LDA # <DIRNAME
STA ICBADR,X

DIRLOOP
JSR GET7
LDA # ">DIRNAME
STA ICBADR+1,X
LDA #$5
STA ICBLLEN,X
LDA #$0
STA ICBLLEN+1,X
LDA #$6
STA ICAUX1,X
LDA #$3
STA ICCOM,X
JSR CIO
BMI CLOSE7
LDA #$0
STA XPTR
LDA #$64
STA SLOT,X
LDA #$65
STA SLOT+1,X
INC XPTR
INC XPTR
JSR GET7
BMI CLOSE7
CMP #"*"+1
BCS ENDIR
JSR CHROUT
JSR GET7
BMI CLOSE7
LDA #$0
STA DIRCOUNT
JSR GET7
BMI CLOSE7
JSR CHROUT
INC DIRCOUNT
LDA DIRCOUNT
CMP #$8
BNE DNOT8
LDA "."
JSR CHROUT
JMP DIRLOOP
CMP #$11
BNE DIRLOOP
LDA #$5
STA TEM
JSR GET7
DEC TEMP
LDA TEMP
BNE THROW5
JMP REDIR
LDX #$70
LDA #$12
STA ICCOM,X
JSR CIO
LDX #$70
LDY ICSTAT,X
RTS
PHA
LDA #$155
JSR CHROUT
PLA
JSR CHROUT
JSR GET7
BMI CLOSE7
```
The main DOS routine calls the Catalog routine to fill the screen with filenames, then puts the cursor on the current filename, waiting for a keypress.

DOS ISR DELITE
ISR OPENEDITOR
ISR DELITE
LDA #1
STA 752
STA 82
LDA #125
JSR CHROUT
JSR CATALOG
ISR DOSMSG
GETNAME LDA SLOT
STA SCR
LDA SLOT+1
STA SCR+1
LDA #0
STA XSLOT
DEC XPTR
DEC XXVR
NAMELP JSR INVNAME
JSR GETAKEY
LDX #1
STX 752
Now that we’ve got a keypress, we look it up in the keypress table, then vector to the appropriate routine. This is the same ML ON-GOTO routine that we’ve used in several places in SpeedScript, including the CONTROL routine.

LDX DOSTABLE
CMP #97
BCC NOPROB
AND #95
STA TEMP
CMP DOSTABLE,X
BEQ FOUNDIT
Dex
Bne FINDIT
Jmp JNAME
FOUNDIT
Dex
Txa
Lda DOSTABLE+1,X
Pha
Lda DOSADR,X
Pha
Rts
The braces surround control characters, some entered with the ESCape key: cursor-left, cursor-right, cursor-up, cursor-down, CTRL-D, ESCape, and CTRL-L.

DOSTABLE .BYTE 15
.DBYTE "\{LEFT]\{RIGHT]\{UP]\{DOWN]\{DR
1U1234 {ESC} [L]"}

DOSADR .WORD DLEFT-1,DRIGH
T-1,DUP-1,DDO
WN-1,DELFIL
1,RENAME-1
.ONELOCK-1,FORMAT-1,DR
IVE-1,DRIVE-1
,DRIVE-1

Move bar cursor left by decrementing slot pointer.

DLEFT JSR INVNAME
LDX XSLOT
BEQ N RANGE
Dex
Jmp RESLOT

Move bar cursor right by incrementing slot pointer.

DRIGHT JSR INVNAME
LDX XSLOT
INX
INX
Cpx XPTR
BCS N RANGE

Store new slot index.

RESLOT STx XSLOT
LDA SLOT,X
STA SCR
LDA SLOT+1,X
STA SCR+1
JMP NAME LP

Move bar cursor up by subtracting 6 from the slot pointer (each slot is two bytes).

DUP JSR INVNAME
LDA XSLOT
CMP #6
Bcc NRANGE
Sec
Sbc #6
Tax
Jmp RESLOT
Move bar cursor down by adding 6 to
the slot pointer.

```
DDOWN                JSR    INVNAME
LDA    XSLOT
CLC
ADC   #6
CMP    XPTR
BCS    N RANGE
TAX
JMP    RESLOT
```

This routine turns a filename pointed to
by the bar cursor into a legal CIO file-
name, complete with Dx: and legal
extension.

```
NAMER            LDX   #0
LDA    DIRNAME,X
STA    FNBUFF,X
INX
CPX   #3
BNE    COPYD
LDY   #1
COPYD
LDA    (SCR),Y
AND   #127
JSR    INTOAS
CMP   #32
BEQ    NOSTOR
STA    FNBUFF,X
INX
NOSTOR
INY
CPY   #13
BNE    COPYNAME
LDA    FNBUFF-1,X
CMP   #.
BNE    NOTDOT
DEX
NOTDOT
STX    FNLEN
LDA   #0
STA    FNBUFF,X
RTS
```

This routine passes any CIO command
along with a formed filename.

```
XIO         LDX   #$70
STA    ICCOM,X
LDA    FNLEN
STA    ICBLEN,X
LDA   #0
STA    ICBLEN+1,X
LDA   #<FNBUFF
STA    IBADR,X
LDA   #>FNBUFF
STA    ICBADR+1,X
JMP    CIO
```

The DOS functions are quite short.
NAMER builds the name; then we sim-
ply pass the number of the DOS CIO
function unto XIO. If there’s no error,
we return to waiting for the next key-
stroke; otherwise, print the DOS error
message and wait for a keystroke.

```
DELFILE        JSR    NAMER
LDA   #33
```

Jump to the XIO routine.

```
GOXIO         JSR    XIO
BPL    JNAME
JMP    DOSERR
JNAME        JSR    INVNAME
JMP    NAMELP
```

Lock a file.

```
LOCK         JSR    NAMER
LDA   #35
JMP    GOXIO
```

Unlock a file.

```
UNLOCK       JSR    NAMER
LDA   #36
JMP    GOXIO
```

We ask for the new name of the file,
built the rename string, then jump to
the XIO routine.

```
RENAME       JSR    BOTCLR
LDA   #<RENMSG
LDY   #>RENMSG
JSR    PRMSG
LDA   #64
STA    $02BE
JSR    INPUT
LDA   #0
STA    $02BE
LDA    INLEN
BEQ    NONAME
JSR    NAMER
LDX   #0
LDY    FNLEN
LDA   #'
STA    FNBUFF,Y
INY
COPYR       LDA    INBUFF,X
STA    FNBUFF,Y
INY
INX
CPX    INLEN
BNE    COPYR
STY    FNLEN
LDA   #0
STA    FNBUFF,Y
JSR    DOSMSG
LDA   #32
JMP    GOXIO
NONAME      JSR    DOSMSG
JMP    JNAME
```

Format routine. We use YORN to af-
firn this operation, which erases an en-
tire disk. BOTCLR clears the bottom
line of the screen.
Select new drive number and redisplay directory.

DRIVE LDA TEMP
STA DIRNAME+1
JMP DOS

The Load-from-directory routine opens the file, then jumps into the SpeedScript Load routine.

LOADIT LDX #$70
STX IOCB
LDA #4
STA ICAUX1,X
LDA #0
STA INDIR
STA INDIR+1
JSR NAMER

Command 3 is for OPEN file.

LDA #3
JSR XIO
BMI DOSERR
JSR ERASE
JSR LOADLINK

If the load ended with an error, we display the error; otherwise, we exit the DOSPAK at ESCDOS.

BMI DOSERR

The ESCape DOS routine clears the stack, clears the screen, reenables the display-list interrupt, prints the “SpeedScript” message, then jumps back to the editing loop.

ESCDOS LDX #$FA
TXS
LDA #125
JSR CHROUT
JSR HIGHLIGHT
JSR SYMSMG
JMP MAIN

BOTCL ERases the bottom two lines of the screen by positioning the cursor on the next-to-the-last line, then printing two INSERT LINE characters that push any text on these lines off the bottom of the screen. Nifty, eh?

BOTCL LDA #22
STA 84
LDA #157
JSR CHROUT
JMP CHROUT

This is the error routine for the DOSPAK. We print “ERROR #”, then print the error number with OUTNUM, a bell character (actually sounds like an annoying buzzer, appropriate Pavlovian treatment), then “Press RETURN.” We wait for a keystroke, then return to getting keys for the DOSPAK commands.

DOSERR STY YSAVE
JSR CLOSE7
JSR BOTCLR
LDA #<ERRMSG
LDY #>ERRMSG
JSR PRMSG
LDX YSAVE
LDA #0
JSR OUTNUM
LDA #253
JSR CHROUT
LDA #<DIRMSG
LDY #>DIRMSG
JSR PRMSG
JSR GETAKEY
JSR DOSMSG
JMP JNAME

Inverse the filename field of the currently selected filename. Used to create the bar cursor.

INVNAME LDA (SCR),Y
EOR #128
STA (SCR),Y
DEY
BMI INVLP
RTS

DOSMSG erases the bottom line of the screen and prints the DOSPAK command line, an abbreviated menu.

DOSMSG JSR BOTCLR
LDA #<DIRINS
LDY #>DIRINS
JSR PRMSG
LDA DIRNAME+1
JMP CHROUT
.END

Filename D:SPEED.2

This is the main input/output portion of SpeedScript, responsible for loading, saving, and all printing functions.
CAST and CINSTOAS (standing for Convert to ASCII and Convert IN- Ternal code to ASCII) translate the way SpeedScript stores text in memory (in- ternal screen codes) into ASCII so that disk files will be compatible with most other software. In addition, the return- mark is changed to character 155, and vice versa. This is why you can’t load a machine language file into SpeedScript, edit it, then save it back as a runnable modification. All back-arrows are turned into carriage returns on output, and all carriage returns (155’s) are turned into back-arrows (30’s) on input.

CAST

<table>
<thead>
<tr>
<th>LDA</th>
<th>#0</th>
</tr>
</thead>
<tbody>
<tr>
<td>STA</td>
<td>CONVFLAG</td>
</tr>
<tr>
<td>JMP</td>
<td>CAST1</td>
</tr>
</tbody>
</table>

CINTOAS

<table>
<thead>
<tr>
<th>LDA</th>
<th>#128</th>
</tr>
</thead>
<tbody>
<tr>
<td>STA</td>
<td>CONVFLAG</td>
</tr>
</tbody>
</table>

CAST1

<table>
<thead>
<tr>
<th>LDA</th>
<th>TEXSTART</th>
</tr>
</thead>
<tbody>
<tr>
<td>STA</td>
<td>TEX</td>
</tr>
<tr>
<td>LDA</td>
<td>TEXSTART+1</td>
</tr>
<tr>
<td>STA</td>
<td>TEX+1</td>
</tr>
<tr>
<td>JMP</td>
<td>CIN</td>
</tr>
</tbody>
</table>

CASTOIN

<table>
<thead>
<tr>
<th>LDA</th>
<th>#0</th>
</tr>
</thead>
<tbody>
<tr>
<td>STA</td>
<td>CONVFLAG</td>
</tr>
<tr>
<td>STA</td>
<td>CURR</td>
</tr>
<tr>
<td>LDA</td>
<td>CURR+1</td>
</tr>
<tr>
<td>STA</td>
<td>TEX+1</td>
</tr>
</tbody>
</table>

CIN

<table>
<thead>
<tr>
<th>SEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA</td>
</tr>
<tr>
<td>SBC</td>
</tr>
</tbody>
</table>

CVLOOP

<table>
<thead>
<tr>
<th>LDA</th>
<th>(TEX),Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>CONVFLAG</td>
</tr>
<tr>
<td>BMI</td>
<td>COTHER</td>
</tr>
<tr>
<td>CMP</td>
<td>#155</td>
</tr>
<tr>
<td>BNE</td>
<td>NOTRC</td>
</tr>
<tr>
<td>LDA</td>
<td>#RETCHAR</td>
</tr>
<tr>
<td>JMP</td>
<td>OVEROTHER</td>
</tr>
</tbody>
</table>

NOTRC

<table>
<thead>
<tr>
<th>JSR</th>
<th>ASTOIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>OVEROTHER</td>
</tr>
</tbody>
</table>

COTHER

<table>
<thead>
<tr>
<th>CMP</th>
<th>#RETCHAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNE</td>
<td>NOTRC</td>
</tr>
<tr>
<td>LDA</td>
<td>#155</td>
</tr>
<tr>
<td>JMP</td>
<td>OVEROTHER</td>
</tr>
</tbody>
</table>

NOTR TN

<table>
<thead>
<tr>
<th>JSR</th>
<th>INTOAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>OVEROTHER</td>
</tr>
</tbody>
</table>

OVEROTHER

<table>
<thead>
<tr>
<th>STA</th>
<th>(TEX),Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>INY</td>
<td></td>
</tr>
<tr>
<td>BNE</td>
<td>CVLOOP</td>
</tr>
<tr>
<td>INC</td>
<td>TEX+1</td>
</tr>
<tr>
<td>DEX</td>
<td></td>
</tr>
<tr>
<td>BNE</td>
<td>CVLOOP</td>
</tr>
<tr>
<td>RTS</td>
<td></td>
</tr>
</tbody>
</table>

Here is where most of the input/output routines start. TSAVE saves the entire document area using the CIO block output routine (PUT TEXT). TOPEN is called by both TSAVE and TLOAD to get the filename and open the file. The device specification (D: or C:) must be typed in by the user.

TSAVE prints the Save: prompt, goes to TOPEN with an 8 (for output, the same number in OPEN 1,8,0,"D:file"), and uses IOC B #7 (LDX #70) to send a PUT TEXT command (11). Text is written from the start-of-text with a length of LASTLINE—TEXSTART.

TSAVE

<table>
<thead>
<tr>
<th>JSR</th>
<th>TOPCLLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA</td>
<td>#&lt;SAVMSG</td>
</tr>
<tr>
<td>LDY</td>
<td>#&gt;SAVMSG</td>
</tr>
<tr>
<td>JSR</td>
<td>PRMSG</td>
</tr>
<tr>
<td>LDA</td>
<td>#8</td>
</tr>
<tr>
<td>JSR</td>
<td>TOPEN</td>
</tr>
<tr>
<td>BMI</td>
<td>ERROR</td>
</tr>
<tr>
<td>JSR</td>
<td>CINTOAS</td>
</tr>
<tr>
<td>LDX</td>
<td>#70</td>
</tr>
<tr>
<td>LDA</td>
<td>TEXSTART</td>
</tr>
<tr>
<td>STA</td>
<td>ICBADR,X</td>
</tr>
<tr>
<td>LDA</td>
<td>TEXSTART+1</td>
</tr>
<tr>
<td>STA</td>
<td>ICBADR+1,X</td>
</tr>
<tr>
<td>SEC</td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>LASTLINE</td>
</tr>
<tr>
<td>SBC</td>
<td>TEXSTART</td>
</tr>
<tr>
<td>STA</td>
<td>ICBLEN,X</td>
</tr>
<tr>
<td>LDA</td>
<td>LASTLINE+1</td>
</tr>
<tr>
<td>SBC</td>
<td>TEXSTART+1</td>
</tr>
<tr>
<td>STA</td>
<td>ICBLEN+1,X</td>
</tr>
<tr>
<td>LDA</td>
<td>#11</td>
</tr>
<tr>
<td>STA</td>
<td>ICCOM,X</td>
</tr>
<tr>
<td>JSR</td>
<td>CIO</td>
</tr>
</tbody>
</table>

The N (negative) bit is set when an error occurs after a call to CIO or a routine that ends up calling CIO. Therefore, we can use BMI to branch on an error condition.

ERR1

<table>
<thead>
<tr>
<th>TYA</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHA</td>
</tr>
<tr>
<td>JSR</td>
</tr>
<tr>
<td>PLA</td>
</tr>
<tr>
<td>TAY</td>
</tr>
</tbody>
</table>

The error routine uses the error number found in the Y register, prints the error message with PRMSG, and the error number with OUTNUM. The open file is closed. If the BREAK key was used to stop the operation, we distinguish this
from an ordinary error, and print "BREAK Abort" instead.

```
ERROR
CPY #128
BEQ STOPPED
TYA
PHA
LDA #125
JSR CHROUT
LDA #<ERRMSG
LDY #>ERRMSG
JSR PRMSG
PLA
TAX
LDA #0
JSR OUTNUM
JSR HIGHLIGHT
LDA #1
STA MSGFLG
RTS
STPPED
JSR TOPCLR
LDA #<BRMSG
LDY #>BRMSG
JSR PRMSG
JMP ERXIT
STPPED
JSR TOPCLR
LDA #<BRMSG
LDY #>BRMSG
JSR PRMSG
JMP ERXIT
```

General file closing routine. IOCB contains the channel number times 16.

```
IOCLOSE
LDX IOCB
LDA #12
STA ICCOM,X
JMP CIO
```

TOPEN is used to get a filename, including the device specification. It's used by Save, Load, and Print. It forces the CAPS key to uppercase for the filename, which is not quite as satisfactory as converting the filename if lowercase was used. It does return the CAPS key to its former value, though. TOPEN opens the file and returns with the error code in the Y register.

```
TOPEN
LDX #$70
STX IOCB
STA ACCESS
JMP CIO
```

Save current CAPS value.
```
LDA SHFLOK
PHA
```

CAPS On.
```
LDA #64
STA SHFLOK
JSR INPUT
```

Restore CAPS value.
```
PLA
STA SHFLOK
LDA INLEN
BNE OPCONT
```

The Load routine checks the cursor position. If the cursor is at the top-of-text (CURR=TEXSTART), we call the ERASE routine to wipe out memory before the load. Otherwise, the load starts at the cursor position, performing an append, and we change the command line to green ($C4, sorry about not using a label) to warn the user. We open the file for reading by passing a 4 to TOPEN, then at LOADLINK use GET TEXT (command 7) to get no more than the length of the text area. The actual length loaded is found in ICBLLEN, so we add this to TEXSTART and the offset between the cursor position and TEXSTART to get the position of the end-of-text (LASTLINE).

A funny thing happens, though. Up to 255 garbage characters appear following an otherwise normal load, after the end-of-text. I was never able to figure out why (and I puzzled over it for a week), so I wrote a stopgap routine to just clear out one page past the end-of-text. The bug is not fixed per se, but it has no effect anymore! I still think it must be the fault of the operating system (I know...).

```
TLOAD
SEC
LDA CURR
SBC TEXSTART
STA TEX
STA INDIR
LDA CURR+1
SBC TEXSTART+1
STA TEX+1
STA INDIR+1
ORA TEX
```
BEQ LOAD2
LDA #$C4
STA WINDCOLR
JSR TOPCLR
LDA #<LOADMSG
LDY #>LOADMSG
JSR PRMSG
LDA #4
JSR TOPEN
BPL OKLOD
JMP ERROR
CMP #$C4
BEQ NOER
JSR ERASE
NOER
JSR LOADLINK
CPY #128
BCC JFINE
JMP ERROR
JMP FINE

Entry point for linked files loading.

LOADLINK LDX IOCB
LDA CURR
STA ICBA0R,X
LDA CURR+1
STA ICBA0R+1,X
SEC
LDA TEXEND
SBC CURR
STA ICBLEN,X
LDA TEXEND+1
SBC CURR+1
STA ICBLEN+1,X
LDA #7
STA ICCOM,X
JSR CIO
BPL TEXOK
CPY #136
BEQ TEXOK
RTS
TEXOK
LDX IOCB
CLC
LDA ICBLEN,X
ADC TEXSTART
STA LASTLINE
LDA ICBLEN+1,X
ADC TEXSTART+1
STA LASTLINE+1
CLC
LDA LASTLINE
ADC INDIR
STA LASTLINE
LDA LASTLINE+1
ADC INDIR+1
STA LASTLINE+1
JSR CASTOIN
LDA LASTLINE
STA TEX+1
LDA #0
TAY

NOGARBAGE STA (TEX),Y
INY
BNE NOGARBAGE
RTS
FINE JSR IOCLOSE
BPL PROKMSG
JMP ERROR
PROKMSG LDA #125
JSR CHROUT
LDA #<OKMSG
LDY #>OKMSG
JSR PRMSG
JMP ERXIT

Disable display-list interrupt and restore screen colors.

DELITE LDA #$40
STA $D40E
LDA SCRCOL
STA 710
STA 712
LDA TEXCOLR
STA 709
RTS

A rather short routine that converts a string of ASCII digits into a number in hex and the accumulator. It takes advantage of decimal mode. In decimal mode, the accumulator is adjusted after additions and subtractions so that it acts like a two-digit decimal counter. We shift BCD over a nybble and add in the left nybble of the ASCII number until we reach the end of the ASCII number. We then subtract 1 from BCD and increment X (which doesn't conform to decimal mode) until BCD is down to 0. The X register magically holds the converted number. Naturally, decimal mode is cleared before this routine exits, or it would wreak major havoc. ASCHEX is used to convert the parameters of printer commands like left margin.

ASCHEX LDX #0
STX BCD
STX BCD+1
STX HEX
STX HEX+1
SEC

DIGIT LDA (TEX),Y
SBC #16
BCC NONUM
CMP #10
BCC NONUM
ASL BCD
ROL BCD+1
ASL BCD
ROL BCD+1
Insert the buffer. This is the recall routine called by CTRL-R. It must not allow an insertion that would overfill memory. It calls DMOVE to open a space in memory, then UMOVE (which is a little faster than DMOVE) to copy the buffer to the empty space.

Exchange the character highlighted by the cursor with the character to the right of it. Not a vital command, but it was included due to the brevity of the code.

Change the case of the character highlighted by the cursor.
Convert internal (screen code) format to Atari ASCII (ATASCII). Used to convert the screen-code format of SpeedScript documents to ASCII for the sake of printing.

The start of the printer routines. This part could logically be called a separate program, but many variables are common to the above code.

DEFTAB: Table of default settings for left margin, right margin, page length, top margin, bottom margin, etc. See the table starting at LMARGIN at the end of this source code.

DEFTAB .BYTE 5,75,66,5,58,1,1,0,0,1,80

Table of default printer codes.

PRCODES .BYTE 27,14,15,18

Another advantage of modular coding is that you can change the behavior of a lot of code by just changing one small common routine. This is a substitute for the normal CHROUT routine. It checks to see if the current page number equals the page number specified by the user to start printing. It also checks for the BREAK to abort the printing and permits printing to be paused with CTRL-1.

PCHROUT STA PCR
TXA
PHA
TYA
PHA

SEC
LDA PAGENUM
SBC STARTNUM
LDA PAGENUM+1
SBC STARTNUM+1
BCC SKIPOUT
LDA #1
STA 766
LDX #$70
LDA #0
STA ICBLLEN,X
STA ICBLLEN+1,X
LDA #11
STA ICOMM,X
LDA PCR
JSR CIO
PHP
LDA #0
STA 766
PLP
BPL SHIFTFREEZE
ERRLINK JSR ERROR
LDX #$FA
TXS
JMP MAIN
SHIFTFREEZE LDA $02FF :CTRL-1
BNE SHIFTFREEZE
SKIPOUT PLA
TAY
PLA
TAX
LOA PCR
RTS

Displays "Printing..."

PRIN JSR TOPCLR
LOA #<PRINMSG
LOY #>PRINMSG
JSR PRMSG
JSR DELITE
LOA #8
JSR TOPEN
BPL PROK
JMP PEXIT

Called by CTRL-P. We get the filename to print to (usually P:, although you can use E: to print to the screen) with ICAUX1 set to 8 for output. We exit on any error. The DELITE routine turns off the display-list interrupt, which might otherwise interfere with output timing.

PRINT JSR TOPCLR
LDA #<FNMSG
LDY #>FNMSG
JSR PRMSG
JSR DELITE
LDA #8
JSR TOPEN
BPL PROK
JMP PEXIT

Reset several flags (footer length, header length, true ASCII, underline mode, and linefeed mode). Notice how DELITE is called again. This isn’t a
mistake. The first time we called DELITE, we then may have opened a file to the Editor device. This reset the screen to the default colors, so the second DELITE retains the user's true color choice.

COPYDEF lda deftab,x sta lmargin,x inx cpd #12 bne copydef lda #$ff sta line sta nomarg ldx #4
COPYDEFS lda prcodes-1,x sta codebuffer+16,x dex bne copydefs

Copy definition tables and default printer codes.

ICEA POS
LDA POS
CMP RMargin
BCC PLOOP1
STY FINPOS
LDA ((TEX),Y
CMP #SPACE
BEQ FOUNDSPACE
DEC POS
DEY
BNE FOUNDSPACE
LDY FINPOS

FOUNDSPACE STA FINPOS
OVERSTOR TYA
SEC
ADC TEX
SVA TEX
LDA TEX+1
ADC #0
STA TEX+1
LDY #0

If this is the first page, we need to print the header, if any, with JSR TOP.

DOBUFF lda line cmp #$ff bne DOBUF2 jsr TOP
LDX line cmp #SPACE beq FOUNDSPACE

DOBUF2 lda nomarg beq OVERMARG jsr LMARG

OVERMARG sec rol nomarg lda finpos sta endpos lda #<PRBUFF sta indir lda #>PRBUFF sta indir+1 jsr BUFPRV

A line has been printed. We check to see if we've hit the bottom margin and, if so, go to PAGE, which goes to the end of the page, prints the footer (if any), and feeds to the next page.

ZBUFF jsr CRFL lda line cmp botmarg bcc NOTPAGE jsr PAGE

Have we reached the end-of-text?

NOTPAGE sec lda TEX sbc lastline
STA TEMP
LDA TEX+1
SBC LASTLINE+1
ORA TEMP
BEQ DORPT
BCC DORPT

If so, we check for a footer. If there is one, we set HOLEN and TOPMARG to 0 (so that the printhead will end up at the right place on the last page) and call PAGE, which prints the footer. If there is no footer, we leave the printhead on the same page so that paper isn’t wasted.

LDA FTLEN
BEQ PXIT
LDA #0
STA HDLEN
STA TOPMARG
JSR PAGE

Exit routines. If screen output was selected, we wait for a keystroke before going back to editing mode.

PXIT LDA INBUFF
CMP #’E
BNE PEXIT
LDA #155
JSR CHROUT
LDA # <DIRMSG
LDY # >DIRMSG
JSR PRMSG
JSR GETAKEY

PEXIT JSR CLOSE7
LDX #$FA
TXS
JSR HIGHLIGHT
LDA #125
JSR CHROUT
JSR SYMSG
JMP MAIN
JMP LOOP

DORPT

SpeedScript Source Code

NEXPAGE BMI NOSK
JSR CR
DEY
BNE NEXPAGE
NOSK LDA FTLEN
BEQ SKIPFT
STA ENDPOS
LDA # <FTBUFF
STA INDIR
LDA # >FTBUFF
STA INDIR+1
JSR LMARG
JSR BUFFRT

SKIPFT JSR CR
JSR CR
JSR CR

Increment the page number.

INC PAGENUM
BNE NOIPN
INC PAGENUM+1

The page wait mode is inappropriate when printing to the screen or to disk, or when skipping over pages with the ? format command.

NOIPN LDA CONTINUOUS
BNE TOP
SEC
LDA PAGENUM
SBC STARTOU
LDA PAGENUM+1
SBC STARTNUM+1
BCC TOP
JSR TOPCLR
LDA # <WAITMSG
LDY # >WAITMSG
JSR PRMSG
JSR GETAKEY
JSR PRIN

Print the header; skip to the top margin.

TOP LDA HDLEN
BEQ NOHEADER
STA ENDPOS
LDA # <HDBUFF
STA INDIR
LDA # >HDBUFF
STA INDIR+1
JSR LMARG
JSR BUFFYZ

NOHEADER LDA TOPMARG
STY LINE
DEY
BEQ SKIPTOP
BMI SKIPTOP

TOPLP JSR CR
DEY
BNE TOPLP

SKITOP RTS

3BC1

3BC1
Left margin routine. This routine is not called if NOMARG is selected (margin release).

```
LMARG  LDA #32
       LDY L_MARGIN
       STY POS
       BEQ LMEXIT

LMLOOP  JSR PCHROUT
        DEY
        BNE LMLOOP

LMEXIT  RTS
```

CRLF is called at the end of most printed lines. It increments the LINE count and takes into account the current line spacing mode set by the s format command.

```
CRLF  LDY SPACING
       CLC
       TYA
       ADC LINE
       STA LINE

CRLOOP  JSR CR
        DEY
        BNE CRLOOP
        RTS
```

CR just prints a single carriage return and linefeed (if specified).

```
CR  LDA #155
       JSR PCHROUT
       BEQ NOLF
       JSR PCHROUT
       RTS
```

NOLF Handle special printer codes like left margin. This looks up the printer code using a routine similar to CONTROL.

```
SPECIAL  STA SAVCHAR
         AND #127
         JSR INTOAS
         LDX SPTAB

SRCHSP  CMP SPTAB,X
         BEQ FSP
         DEX
         BNE SRCHSP
         DEC POS
         JMP DEFINE

FSP  DEX
      TXA
      ASL A
      TAX
      STY YSAVE
      LDA # >SPCONT-1
      PHA
      LDA # <SPCONT-1
      PHA
      LDA SPVECT+1,X
      PHA
```

After the format code is processed, we must skip over the format command and its parameter so that it’s not printed.

```
SPCONT  SEC
        LDA YSAVE
        ADC TEX
        STA TEX
        LDA TEX+1
        ADC #0
        STA TEX+1
        JMP PLOOP
```

If the format command ends with a return-mark, we must skip over the return-mark as well.

```
SPCEXIT  LDA (TEX),Y
         CMP #RETCHAR
         BEQ NOAD
         DEY
         NOAD  STY YSAVE
         RTS
```

Special format code table. It starts with the number of format commands, then the characters for each format command.

```
SPTAB .BYTE 17
          "wlrtbshf@p?xmgj"
```

The address−1 of each format routine.

```
SPVECT .WORD PW-1,LM-1,RM-1,T TEMP-1
         .WORD BT-1,SP-1,NX-1,HD-1,FT-1
         .WORD PN-1,PL-1,SPAGE-1
         .WORD ACROSS-1
         .WORD MRELEASE-1,COMME NT-1,LINK-1
         .WORD LFSET-1
```

m Margin release. INY is used to skip over the format character.

```
MRELEASE  INY
          LDA #0
          STA NOMARG
          JMP SPCEXIT
```

x Columns across, used by centering.

```
ACROSS  INY
        JSR ASCHEX
        STA PAGEWIDTH
        JMP SPCEXIT
```

? Start printing at specified page.

```
SPAGE  INY
        JSR ASCHEX
```

100
STA STARTNUM
LDA HEX+1
STA STARTNUM+1
JMP SPCEXIT

@ Set starting default page number.

PN
INY
JSR ASCHEX
STA PAGENUM
LDA HEX+1
STA PAGENUM+1
JMP SPCEXIT

p Page length.

PL
INY
JSR ASCHEX
STA PAGENUM
LDA HEX+1
STA PAGENUM+1
JMP SPCEXIT

w Set page wait mode.

PW
LDA #0
STA CONTINUOUS
INY
JMP SPCEXIT

j Set linefeed mode.

LFSET
LDA #10
STA LINEFEED
INY
JMP SPCEXIT

l Left margin.

LM
INY
JSR ASCHEX
STA LMMARGIN
JMP SPCEXIT

r Right margin.

RM
INY
JSR ASCHEX
STA RMMARGIN
JMP SPCEXIT

t Top margin.

TP
INY
JSR ASCHEX
STA TOPMARG
JMP SPCEXIT

b Bottom margin.

BT
INY
JSR ASCHEX
STA BOTMARG
JMP SPCEXIT

s Set line spacing.

SP
INY
JSR ASCHEX
STA SPACING
JMP SPCEXIT

n Jump to next page.

NX
LDY YSAVE
INY
TYA
PHA
JSR PAGE
PLA
TAY
STY YSAVE
RTS

h Define header. Copy header into header buffer.

HD
JSR PASTRET
DEY
STY HDLEN
LDY #1
HDCOPY
LDA (TEX),Y
STA HDBUFF-1,Y
INY
CPY HDLEN
BCC HDCOPY
BEQ HDCOPY
INY
JMP SPCEXIT

Skip just past the return-mark.

PASTRET
INY
LDA (TEX),Y
CMP #RETCHAR
BNE PASTRET
RTS

f Define footer.

FT
JSR PASTRET
DEY
STY FTLEN
LDY #1
FTCOPY
LDA (TEX),Y
STA FTBUFF-1,Y
INY
CPY FTLEN
BCC FTCOPY
BEQ FTCOPY
JMP SPCEXIT

i Ignore a line of information.

COMMENT
JSR PASTRET
JMP SPCEXIT

Define programmable printkeys. We check for =. If not found, this is not an assignment, so we just skip past the code. Otherwise, we use the screen code value as the index into the CODEBUFFER and put the value there, ready to be called during printing by BUFPT.

DEFINE
INY
LDA (TEX),Y
SpeedScript

CMP  #* = -32
BEQ  DODF
DEY
LDA  SAVCHAR
JMP  NOTRET
INY
JSR  ASCHEX
PHA
LDA  SAVCHAR
AND  #127
TAX
PLA
STA  CODEBUFFER,X
JSR  SPCEXIT
JMP  SPCONT

DODF

g Link to next file. We get the filename from text and put it into the input buffer, just as if the filename were typed in with INPUT. We then jump into the TOPEN routine to open the file, and into the Load routine to load the file. After the load, we check for a load error; then jump to RETEX to continue printing.

LINK
LDY  #1
LDX  #0

FNCOPY
LDA (TEX),Y
CMP #RETCCHAR
BEQ  FNEND
JSR  INTOAS
STA  INBUFF,X
INY
INX
CPX  #14
BNE  FNCOPY

FNEND
STX  INLEN
LDA  #0
STA  INBUFF,X
LDX  #$60
STX  IOC
LDA  #0
STA  ACCESS
JSR  OPCONT
BPL  LNOERR
JMP  ERRLINK

LNOERR
LDA  #0
STA  INDIR
STA  INDIR+1
JSR  ERASE
JSR  LOADLINK
BPL  LCONT
JMP  ERRLINK

LCONT
PLA
PLA
LDX  #$70
STA  IOC
JMP  RETEX

Global search and replace. This just links together the search-specify routine, the replace-specify routine, then repeatedly calls Hunt and Replace, until Hunt returns “Not Found.” (FPOS+1 is $FF after a search failure.)

SANDR
JSR  RESET
LDA  HUNTLEN
BEQ  NOSSR
JSR  ASKREP
SNR
JSR  CONTSRCH
LDA  FPOS+1
CMP  #$FF
BEQ  NOSSR
JSR  REPL
JSR  REFRESH
JMP  SNR

NOSSR
JMP  SYSMSG

If OPTION is held down with CTRL-F, we ask for and store the search phrase. If OPTION is not down, we perform the actual search. The line in the INBUFF is compared with characters in text. If at any point the search fails, we continue the comparison with the first character of INBUFF. The search is a failure if we reach the end-of-text. If the entire length of INBUFF matches, the search succeeds, so we change the CURRENT cursor position to the found position, save the found position for the sake of the replace routine, then call CHECK to scroll to the found position.

HUNT
LDA  #8
STA  53279
LDA  53279
CMP  #3
BNE  CONTESRCH
JSR  TOPCLR

RESET
JSR  HUNTLEN
BNE  OKSRCH
JMP  SYSMSG

OKSRCH
LDY  #0

OKSRCH
LDY  #0

TOBUFF
LDA  INBUFF,Y
STA  HUNTBUFF,Y
INY
CPY  INLEN
BNE  TOBUFF
JMP  SYSMSG

CONTESRCH
LDA  CURR
STA  TEX
LDA  CURR+1
STA  TEX+1
LDX  #$70
STA  FPOS+1
LDY  #1
LDX  #0

SRCH0
LDX  #0

Global search and replace. This just links together the search-specify routine, the replace-specify routine, then repeatedly calls Hunt and Replace, until Hunt returns “Not Found.” (FPOS+1 is $FF after a search failure.)

SANDR
JSR  RESET
LDA  HUNTLEN
BEQ  NOSSR
JSR  ASKREP
SNR
JSR  CONTSRCH
LDA  FPOS+1
CMP  #$FF
BEQ  NOSSR
JSR  REPL
JSR  REFRESH
JMP  SNR

NOSSR
JMP  SYSMSG

If OPTION is held down with CTRL-F, we ask for and store the search phrase. If OPTION is not down, we perform the actual search. The line in the INBUFF is compared with characters in text. If at any point the search fails, we continue the comparison with the first character of INBUFF. The search is a failure if we reach the end-of-text. If the entire length of INBUFF matches, the search succeeds, so we change the CURRENT cursor position to the found position, save the found position for the sake of the replace routine, then call CHECK to scroll to the found position.

HUNT
LDA  #8
STA  53279
LDA  53279
CMP  #3
BNE  CONTESRCH
JSR  TOPCLR

RESET
JSR  HUNTLEN
BNE  OKSRCH
JMP  SYSMSG

OKSRCH
LDY  #0

OKSRCH
LDY  #0

TOBUFF
LDA  INBUFF,Y
STA  HUNTBUFF,Y
INY
CPY  INLEN
BNE  TOBUFF
JMP  SYSMSG

CONTESRCH
LDA  CURR
STA  TEX
LDA  CURR+1
STA  TEX+1
LDX  #$70
STA  FPOS+1
LDY  #1
LDX  #0

SRCH0
LDX  #0
LDA HUNTLEN
BEQ NOTFOUND
LDA HUNTBUFF,X
JSR ASTOIN
CMP (TEX),Y
BEQ CY
CPX #0
BNE SRCH0
DEX
INY
BNE NOVFL
INC TEX+1
LDA TEX+1
CMP LASTLINE+1
BEQ NOVFL
BCS NOTFOUND
NOVFL
INX
CPX HUNTLEN
BNE SRCH1
CLC
TYA
ADC TEX
STA TEMP
LDA TEX+1
ADC #0
STA TEMP+1
LDA LASTLINE
CMP TEMP
LDA LASTLINE+1
SBC TEMP+1
BCC NOTFOUND
SEC
LDA TEMP
SBC HUNTLEN
STA CURR
STA FPOS
LDA TEMP+1
SBC #0
STA CURR+1
STA FPOS+1
JSR CHECK
RTS
NOTFOUND
JSR TOPCLR
LDA #<NFMSG
LDY #>NFMSG
JSR PRMSG
LDA #1
STA MSGFLG
RTS

RESTART
LDA #8
STA 53279
LDA 53279
CMP #3
BNE REPL
ASKREP
JSR TOPCLR
LDA #<REPMSG
LDY #>REPMSG
JSR PRMSG
JSR INPUT
STA REPLLEN
BEQ NOREP
LDY #0
REPMOV
LDA INBUFF,Y
STA REPBUFF,Y
INY
CPY INLEN
BNE REPMOV
NOREP
JMP SYMSG
REPL
SEC
LDA CURR
STA DESTL
SBC FPOS
STA TEMP
LDA CURR+1
STA DESTH
SBC FPOS+1
ORA TEMP
BNE NOREPL
LDA #$FF
STA FPOS+1
LDA HUNTLEN
ADC CURR
STA FROML
LDA #0
ADC CURR+1
STA FROMH
SEC
LDA LASTLINE
SBC DESTL
STA LLEN
LDA LASTLINE+1
SBC DESTH
STA HLEN
JSR UMOVE
SEC
LDA LASTLINE
SBC HUNTLEN
STA LASTLINE
LDA LASTLINE+1
SBC #0
STA LASTLINE+1
LDA REPLLEN
BEQ NOREP
STA INSLEN
LDA #0
STA INSLEN+1
JSR INSBLOCK
LDY #0
REPLOOP
LDA REPLBUFF,Y
JSR ASTOIN
STA (CURR),Y

The change (replace) routine checks to see if OPTION is held down with CTRL-C. If it is, we ask for a replace phrase, and exit. If not, we check to see if the cursor is at the position previously located by the search routine. If it is, we delete the found phrase, then insert the replace phrase. The cursor is moved past the replace phrase for the sake of the next search. This also prevents endless recursion, as in replacing in with winner.
Suddenly, we're back to a PRINT subroutine. This examines the buffer as it's being printed, checking for printkeys and Stage 2 commands like centering.

In underline mode, after we print the character, we backspace the printhead and print an underline character.

Stage 2 format commands.

Do special format codes. This just uses the screen-code value of the character as an index into the CODEBUFFER, then sends out the code. SpeedScript makes no judgment on the code being sent out.

Display free memory using OUTNUM.

104
Messages are stored in ATASCII, with a zero byte for a delimiter.

```
MG1 .BYTE "SpeedScript 3.0" .BYTE 0
MG2 .BYTE "by Charles Brannon" .BYTE 0
KILL .BYTE "Buffer Cleared" .BYTE 0
BUFERR .BYTE "Buffer Full" .BYTE 0
DELM .BYTE "Delete (S,W,P)" .BYTE 0
YMS .BYTE ": Are you sure? (Y/N): " .BYTE 0
CLRM .BYTE "ERASE ALL TEXT" .BYTE 0
ERAM .BYTE "Erase (S,W,P): TURN to exit" .BYTE 0
SAVMS .BYTE "Save (Device:Filename)" .BYTE 0
ERRM .BYTE "Error #" .BYTE 0
BRM .BYTE "BREAK Key Abort" .BYTE 0
OKMS .BYTE "No Errors" .BYTE 0
LOADMS .BYTE "Load (Device:Filename)" .BYTE 0
DIRMS .BYTE "Press RETURN" .BYTE 0
DIRN .BYTE "DI: *. " .BYTE 0
INSE .BYTE "Memory Full" .BYTE 0
INSMS .BYTE "No text in buffer" .BYTE 0
FM .BYTE "Print (Device:Filename): " .BYTE 0
PRINMS .BYTE "Printing... " .BYTE 0
WAITMS .BYTE "Insert next sheet, press RETURN" .BYTE 0
SRCHMS .BYTE "Find:" .BYTE 0
NFMS .BYTE "Not found" .BYTE 0
REPM .BYTE "Change to:" .BYTE 0

The {ESC}'s represent the ESCape key. The arrows are the cursor keys, which must be preceded by ESC to be entered into text. There is actually only one space between the e of Rename and the E of ESC.

```

The .OPT NO OBJ and .OPT OBJ pseudo-ops turn on and off object code generation. This insures that no object code is generated for the variable table.

```

```
FTLEN *= +1 ;Footer length
LMARGIN *= +1 ;Left margin
RMARGIN *= +1 ;Right margin
PAGELENGTH *= +1 ;Page length
TOPMARG *= +1 ;Top margin
BOTMARG *= +1 ;Bottom margin
SPACING CONTINUOUS *= +1 ;Line spacing
PAGEWAIT *= +1 ;Page wait mode
NOMARG *= +1 ;Margin release flag
POS *= +1 ;Position within line
LINE *= +1 ;Line count
YSAVE *= +1 ;Y register
SAVCHAR *= +1 ;Preserves accumulator
INSLEN *= +1 ;Length of an insertion
DEVNO *= +1 ;Device number
NEEDASC *= +1 ;True ASCII flag
UNDERLINE *= +1 ;Underline mode flag
FPOS *= +2 ;Found position
PCR *= +1 ;Used by PCHROUT
HUNTLLEN *= +1 ;Length of hunt phrase
HUNTBUFF *= +30 ;Holds hunt phrase
REPLEN *= +1 ;Length of replace phrase
REPBUFF *= +30 ;Holds replace phrase
CODEBUFFER *= +128 ;Holds definable printkeys
PRBUFF *= +256 ;Printer line buffer
HDBUFF *= +256 ;Holds header
FIRSTRUN *= +1 ;Has program been run before?
FTBUFF *= +256 ;Holds footer
SAVCOL *= +1 ;Save
LINEFEED *= +1 ;Linefeed mode flag
ESCFLAG *= +1 ;Was ESC pressed?
CONVFLAG *= +1 ;Used by CAST and CINTOAS
SELFAG *= +1 ;The SELECT key flag
IOCB *= +1 ;Which IOCB is OPEN
ACCESS *= +1 ;Direction of ACCESS (read/write)
FNBUFF *= +40 ;Filename buffer
FNLEN *= +1 ;Filename length
XSLOT *= +1 ;Number of filename slots (DOSPAK)
SLOT *= +130 ;Slot positions (DOSPAK)
XPTR *= +1 ;Current filename (DOSPAK)
WHICHFLAG *= +1 ;Which key is pressed
DIRCOUNT *= +1 ;Directory count
BLINK *= +1 ;Cursor blink flag
LINELEN *= +1 ;Length of screen lines
RLM *= +1 ;REFRESH left margin value
KEYVAL *= +1 ;Which key is pressed
END = * ;High byte of this $100 is TEXSTART

Autorun vector

$, 02E2
.WORD "BEGIN"
.END

Label Cross Reference. This chart makes it easier to find your place in the object code while looking at the source code. The number to the left of each label is its value or position within the object code. Labels preceded by an = mark are equates. Others are internal labels for object code positions.
<table>
<thead>
<tr>
<th>SpeedScript Source Code</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3614 ASCHEX</td>
<td>3282 DDOWN</td>
</tr>
<tr>
<td>3C3C ASKREP</td>
<td>3654 DECHEX</td>
</tr>
<tr>
<td>262C ASTOIN</td>
<td>3AF6 DEFINE</td>
</tr>
<tr>
<td>3FCD BCD</td>
<td>3752 DEFTAB</td>
</tr>
<tr>
<td>1F00 BEGIN</td>
<td>2A6C DEL1</td>
</tr>
<tr>
<td>464A BLINK</td>
<td>2A80 DEL1A</td>
</tr>
<tr>
<td>3029 BLIP</td>
<td>2A89 DEL2</td>
</tr>
<tr>
<td>= 0074 BLUE</td>
<td>2A7D DELABORT</td>
</tr>
<tr>
<td>2983 BORDER</td>
<td>2AA0 DELC</td>
</tr>
<tr>
<td>33AA BOTCLR</td>
<td>2B32 DELCHAR</td>
</tr>
<tr>
<td>3FE1 BOTMARG</td>
<td>2B5C DELETE</td>
</tr>
<tr>
<td>3F79 BOTSCR</td>
<td>32E5 DELFILE</td>
</tr>
<tr>
<td>24D3 BREAK</td>
<td>2B4D DELIN</td>
</tr>
<tr>
<td>3E3A BRMSG</td>
<td>35FF DELITE</td>
</tr>
<tr>
<td>3A92 BT</td>
<td>3DC7 DELMSG</td>
</tr>
<tr>
<td>3F6C BUFE ND</td>
<td>2B6A DELENT</td>
</tr>
<tr>
<td>3DBB BUFE RR</td>
<td>2B7D DELWORD</td>
</tr>
<tr>
<td>3FD3 BUFE LN</td>
<td>2B83 DESTH</td>
</tr>
<tr>
<td>3CDE BUFE LP</td>
<td>2B82 DESTL</td>
</tr>
<tr>
<td>3CDC BUFE PR T</td>
<td>3FDD DESTSAV</td>
</tr>
<tr>
<td>33FF CAST</td>
<td>3FEF DEVNO</td>
</tr>
<tr>
<td>340C CAST1</td>
<td>3622 DIGIT</td>
</tr>
<tr>
<td>3419 CASTOIN</td>
<td>4469 DIRCOUNT</td>
</tr>
<tr>
<td>3100 CATALOG</td>
<td>3F00 DIRINS</td>
</tr>
<tr>
<td>27CF CHECK</td>
<td>3156 DIRLOOP</td>
</tr>
<tr>
<td>282C CHECK2</td>
<td>3E5C DIRMSG</td>
</tr>
<tr>
<td>2F7F CHROUTE</td>
<td>3E7A DIRNAME</td>
</tr>
<tr>
<td>2F CB CHRSAVE</td>
<td>25E9 DISKBOOT</td>
</tr>
<tr>
<td>3426 CIN</td>
<td>3247 DLEFT</td>
</tr>
<tr>
<td>3407 CINTOAS</td>
<td>2E0A DLI</td>
</tr>
<tr>
<td>= E456 CIO</td>
<td>2DDE DLIST</td>
</tr>
<tr>
<td>284C CK3</td>
<td>2DC8 DLOOP</td>
</tr>
<tr>
<td>2D02 CLEAR</td>
<td>247A DMOV1</td>
</tr>
<tr>
<td>2501 CLEARED</td>
<td>244D DMOVE</td>
</tr>
<tr>
<td>28E2 CLEFT</td>
<td>2476 DMOVLOOP</td>
</tr>
<tr>
<td>3D1F CLOOP</td>
<td>3170 DNOT8</td>
</tr>
<tr>
<td>3184 CLOSE7</td>
<td>315B DNOTCR</td>
</tr>
<tr>
<td>254C CLR2</td>
<td>3873 DOBUF2</td>
</tr>
<tr>
<td>24F4 CRLRN</td>
<td>3869 DOBUFF</td>
</tr>
<tr>
<td>2543 CRL LOOP</td>
<td>3D62 DOCODES</td>
</tr>
<tr>
<td>3DED CLRM SG</td>
<td>3B04 DODEFINE</td>
</tr>
<tr>
<td>4033 CODEBUFFER</td>
<td>26F4 DOINS</td>
</tr>
<tr>
<td>3A03 COMMENT</td>
<td>2D25 DOTT</td>
</tr>
<tr>
<td>3FE3 CONTINUOUS</td>
<td>3675 DONENUM</td>
</tr>
<tr>
<td>2732 CONTROL</td>
<td>3D50 DOPGN</td>
</tr>
<tr>
<td>38 B7 CONT SRCH</td>
<td>38EB DORPT</td>
</tr>
<tr>
<td>43 B7 CONVFLAG</td>
<td>31BB DOS</td>
</tr>
<tr>
<td>24 D7 COPY</td>
<td>3229 DOASDR</td>
</tr>
<tr>
<td>3296 COPYD</td>
<td>336 DOSERR</td>
</tr>
<tr>
<td>37 EC COPYDEF</td>
<td>33EF DOSMSG</td>
</tr>
<tr>
<td>3801 COPYDEFS</td>
<td>3219 DOSTABLE</td>
</tr>
<tr>
<td>32 A3 COPY NAME</td>
<td>3254 DRI GHT</td>
</tr>
<tr>
<td>3332 COPYR</td>
<td>336F DRIVE</td>
</tr>
<tr>
<td>3446 COTHER</td>
<td>3271 DUP</td>
</tr>
<tr>
<td>3998 CR</td>
<td>2BD9 EATSPACE</td>
</tr>
<tr>
<td>28 AF CRIGHT</td>
<td>3D2F EDGE</td>
</tr>
<tr>
<td>3986 CRLF</td>
<td>2F99 ENAME</td>
</tr>
<tr>
<td>3991 CRL LOOP</td>
<td>446E END</td>
</tr>
<tr>
<td>2753 CTBL</td>
<td>3D00 ENDBUFF</td>
</tr>
<tr>
<td>= 0086 CURR</td>
<td>3194 ENDIR</td>
</tr>
<tr>
<td>2EE7 CURSIN</td>
<td>319 E EN DLP</td>
</tr>
<tr>
<td>3430 CV LOOP</td>
<td>3F73 ENDP O S</td>
</tr>
<tr>
<td>3BDC CY</td>
<td>2967 EN DTEX</td>
</tr>
<tr>
<td>SpeedScript Source Code</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td></td>
</tr>
</tbody>
</table>

<p>| 446B | LINELEN |
| 3B19 | LINK |
| 3A74 | LM |
| 3975 | LMarg |
| 3FDD | LMargin |
| 3985 | LExit |
| 397F | LLoop |
| 3B49 | Load2 |
| 3554 | Loadit |
| 3377 | LoadLink |
| 3E54 | LoadMsg |
| 32F8 | Lock |
| 2C4C | Lottaspace |
| 2645 | Lowr |
| 2648 | Main |
| 2651 | Main2 |
| 2428 | Mov1 |
| 242A | Mov2 |
| 242F | Movloop |
| 3A25 | Mrelease |
| 3D88 | Msg1 |
| 3D98 | Msg2 |
| 3F71 | MsgFlag |
| 31EB | NameF |
| 3294 | NameR |
| 3FE0 | NeedASC |
| 38FC | Nexpage |
| 3EEB | Nfmsg |
| 39ED | Noad |
| 2F1B | Noback |
| 28AC | Nobigger |
| 2583 | Noblink |
| 3CFC | Nobrk |
| 28E8 | Nodec |
| 3571 | Noer |
| 2F04 | Noesc |
| 2A37 | Nofixcurr |
| 3EE2 | NoGarbage |
| 3963 | Noheader |
| 3675 | Nohexinc |
| 272C | Noinc2 |
| 28B5 | Noincr |
| 2C63 | Noincy |
| 3929 | Nopn |
| 39A5 | Nolf |
| 3FE9 | Nomarg |
| 26B8 | Noms |
| 33F4 | Noname |
| 3653 | Nonum |
| 2F91 | Nopar |
| 31FF | Noprob |
| 3C5C | Norep |
| 3CD9 | Norepl |
| 3902 | Nosk |
| 3B82 | Nosr |
| 32B2 | Nostor |
| 3735 | NotAlpha |
| 26AC | Notbk |
| 300F | Notcaps |
| 3D2B | Notcenter |
| 26C2 | Notcr |
| 2642 | Notctrl |
| 32BF | Nodot |
| 3D40 | Notedge |
| 3C20 | Nofound |
| 26F7 | Notinst |
| 3024 | Notlocked |
| 2455 | Notnull |
| 3D4C | Notog |
| 389E | Notpage |
| 2BA0 | Ntpar |
| 344F | Notrc |
| 382F | Notret |
| 3440 | Notrtn |
| 26A6 | Notsel |
| 2B93 | Notsent |
| 3004 | Notset |
| 382B | Notsp |
| 2B86 | Notword |
| 2F13 | Notzero |
| 3BEA | Novfl |
| 2E77 | Noword |
| 326E | Nrange |
| 3AA6 | Nx |
| 24C7 | Nxcur |
| 2941 | Oids |
| 27FF | Ok1 |
| 282C | Ok2 |
| 36A3 | Okbuff |
| 2D0F | Okclear |
| 2CA9 | Okins |
| 3568 | Oklod |
| 36DF | Okmov |
| 3EA4 | Okmsg |
| 3BA6 | Oksrch |
| 27C6 | Onoff |
| 2FCA | Onumexit |
| 2FB1 | Onumloop |
| 3508 | Opabort |
| 3510 | Opcont |
| 2F59 | Openeditor |
| 3D0C | Other |
| 244C | Out |
| 2BC9 | Outhome |
| 2FA0 | Outnum |
| 287A | Outrange |
| 2C0B | Outspace |
| 26E2 | Overctrl |
| 387B | Overmarg |
| 3452 | Overother |
| 2FC4 | Overpchr |
| 385B | Overstor |
| 2FA2 | Overzap |
| 38EE | Page |
| 3FDF | Pagelen |
| 3FE4 | Pagenum |
| 3FE8 | Pagewidth |
| 2D64 | Parcont |
| 2D31 | Parleft |
| 2D52 | Parloop |
| 2D33 | Parlp |
| 3ACF | Pastret |
| 37BB | Pabort |</p>
<table>
<thead>
<tr>
<th>Code</th>
<th>Symbol</th>
<th>Code</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>396E</td>
<td>TOPLP</td>
<td>0091</td>
<td>WINDCOLR</td>
</tr>
<tr>
<td>3FE0</td>
<td>TOPMARG</td>
<td>28ED</td>
<td>WLEFT</td>
</tr>
<tr>
<td>3A88</td>
<td>TP</td>
<td>2906</td>
<td>WLOOP</td>
</tr>
<tr>
<td>3FD1</td>
<td>TPRTR</td>
<td>2923</td>
<td>WRIGHT</td>
</tr>
<tr>
<td>345D</td>
<td>TSAVE</td>
<td>2914</td>
<td>WROUT</td>
</tr>
<tr>
<td>2410</td>
<td>UMOVE</td>
<td>2957</td>
<td>WRTN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0090</td>
<td>UNDERCURS</td>
</tr>
<tr>
<td>3FF1</td>
<td>UNDERLINE</td>
<td>374F</td>
<td>XINT</td>
</tr>
<tr>
<td>3300</td>
<td>UNLOCK</td>
<td>32C8</td>
<td>XIO</td>
</tr>
<tr>
<td>2E94</td>
<td>UNSENT</td>
<td>4467</td>
<td>XPTR</td>
</tr>
<tr>
<td>2777</td>
<td>VECT</td>
<td>43E4</td>
<td>XSLOT</td>
</tr>
<tr>
<td>2667</td>
<td>WAIT</td>
<td>3DD6</td>
<td>YMSG</td>
</tr>
<tr>
<td>3EC5</td>
<td>WAITMSG</td>
<td>2CEB</td>
<td>YORN</td>
</tr>
<tr>
<td>2BBC</td>
<td>WAITST</td>
<td>2CF2</td>
<td>YORNKEY</td>
</tr>
<tr>
<td>4468</td>
<td>WHICHFLAG</td>
<td>3FEC</td>
<td>YSAVE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3890</td>
<td>ZBUFF</td>
</tr>
</tbody>
</table>
Index

ASCII files 13
ASCII value, use of in defining a
printkey 21–22
Atari DOS 2.0, use of 4, 15
Atari DOS 3.0, use of 4, 15
"The Automatic Proofreader" 34–36
preparing the program 34
program listing 35–36
using the program 35
command line 5
control key commands 7–15
CTRL-A 14
CTRL-B 14
CTRL-C 11–12
CTRL-CLEAR 7
CTRL-D 9, 10
CTRL-DELETE/BACK S 9
CTRL-E 9, 10
CTRL-F 11
CTRL-G 11, 12
CTRL-I 8
CTRL-INSERT 8
CTRL-K 10
CTRL-L 11–12
CTRL-M 13
CTRL-O 14
CTRL-P 16
CTRL-R 10
CTRL-S 12
CTRL-T 14
CTRL-U 6
CTRL-X 14
CTRL-Z 8, 13
CTRL-· 7
CTRL-+ 7
CTRL-=` 7
CTRL-=- 7
ESC key, use of with CTRL 8
explanation of how to enter 7
OPTION button, use of with CTRL 7, 11, 14
cursor movement 7–8
cursor-down key 7
cursor-left key 7
cursor-right key 7
cursor-up key 7
delete mode 9–10
disk commands 13–14
entering of text 5–9
erase mode 9
erasing of text 9–10
format commands 15–23
SELECT key 16–17, 21
Stage 1 commands 16–20
Stage 2 commands 20–21
summary of 19
insert mode 8–9
keyboard commands 7–12. See also
c control key commands; keyboard
map, illustration of
BREAK key 16
CAPS/LOWR key 15
DELETE/BACK S key 9
OPTION-CTRL-C 11
OPTION-CTRL-F 11
OPTION-CTRL-G 11
OPTION-CTRL-+ 14
OPTION-CTRL-* 14
SELECT key 16–17, 21
SHIFT-DELETE/BACK S 9
SHIFT-INSERT 8
SHIFT-+ 7
SHIFT-* 7
SHIFT-= 7
START button 8
SYSTEM RESET button 15
TAB key 8
keyboard map, illustration of 18
loading of a document 12
loading of program
binary disk file 4
boot tape 4–5
"The Machine Language Editor: MLX" 27–33
commands 4, 28–29
ending address 3, 27, 37
explanation of use 27
program listing 29–33
run/init address 3, 27, 37
starting address 3, 27, 37
typing in multiple sittings 4–5
MAC/65 Assembler 69
merging, of files 13
Optimized Systems Software, Inc. 5, 69
OPTION button, use of 11
OS/A+ DOS, use of 4, 15
overscanning 14
parsing 6
printing of documents
default settings 16
explanation of use 15–23
to an RS-232 printer 16
printkey, defining of 21–23
program listing 37–65
return-mark 7
saving of the program
as a binary file 3, 27
as a boot tape 3, 27
screen formatting 6–7
scrolling 6–7
search and replace 11–12
source code 69–111
  explanation of 69–70
  listing of 71–111

start a new document 11
storing of a document 12
text buffer 10
typing it in 3–5
  in multiple sittings 4–5
using the program 3–23
width of the screen, how to change it
  14
word wrap 6
To order your copy of the Atari Speedscript Disk call our toll-free US order line: 1-800-346-6767 (in NY 212-887-8525) or send your prepaid order to:

Atari Speedscript Disk
COMPUTE! Publications
P.O. Box 5038
F.D.R. Station
New York, NY 10150

All orders must be prepaid (check, charge, or money order), NC residents add 4.5% sales tax.

Send ____ copies of the Atari Speedscript Disk at $12.95 per copy.

Subtotal $___________

Shipping and Handling: $2.00/disk $___________
($5.00 airmail)

Sales tax (if applicable) $___________

Total payment enclosed $___________

☐ Payment enclosed
☐ Charge ☐ Visa ☐ MasterCard ☐ American Express

Acct. No. ____________________________ Exp. Date ____________
(Required)

Name ________________________________

Address ______________________________

City __________________ State _______ Zip _______

Please allow 4-5 weeks for delivery.
If you've enjoyed the articles in this book, you'll find the same style and quality in every monthly issue of COMPUTE! Magazine. Use this form to order your subscription to COMPUTE!.

For Fastest Service
Call Our Toll-Free US Order Line
1-800-247-5470
In IA call 1-800-532-1272

COMPUTE!
P.O. Box 10954
Des Moines, IA 50340

My computer is:
☐ Commodore 64 or 128 ☐ TI-99/4A ☐ IBM PC or PCjr ☐ VIC-20
☐ Apple ☐ Atari ☐ Amiga ☐ Other
☐ Don't yet have one...

☐ $24 One Year US Subscription
☐ $45 Two Year US Subscription
☐ $65 Three Year US Subscription

Subscription rates outside the US:
☐ $30 Canada and Foreign Surface Mail
☐ $65 Foreign Air Delivery

Name

Address

City State Zip

Country

Payment must be in US funds drawn on a US bank, international money order, or charge card.

☐ Payment Enclosed ☐ Visa
☐ MasterCard ☐ American Express

Acct. No. Expires /

(Required)

Your subscription will begin with the next available issue. Please allow 4–6 weeks for delivery of first issue. Subscription prices subject to change at any time.
<table>
<thead>
<tr>
<th>Quantity</th>
<th>Title</th>
<th>Price</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>COMPUTE!'s First Book of Atari (00-0)</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTE!'s Second Book of Atari (06-X)</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTE!'s Third Book of Atari (18-3)</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTE!'s First Book of Atari Graphics (08-6)</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTE!'s Second Book of Atari Graphics (28-0)</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTE!'s First Book of Atari Games (14-0)</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTE!'s Atari Collection, Volume 1 (79-5)</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTE!'s Atari Collection, Volume 2 (029-7)</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Machine Language for Beginners (11-6)</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Second Book of Machine Language (53-1)</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SpeedScript: The Word Processor for the Atari (003)</td>
<td>$9.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mapping The Atari, Revised (004)</td>
<td>$16.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTE!'s ST Programmer's Guide (023-8)</td>
<td>$16.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Elementary Atari ST (024)</td>
<td>$16.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTE!'s Kids and the ST (386)</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Elementary ST BASIC (343)</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Introduction to Sound and Graphics on the Atari ST (035)</td>
<td>$14.95</td>
<td></td>
</tr>
</tbody>
</table>

Add $2.00 per book shipping and handling. Outside US add $5.00 air mail or $2.00 surface mail.

**NC residents add 4.5% sales tax**

Shipping & handling

Total payment

All orders must be prepaid (money order, check, or charge). All payments must be in US funds.

Payment enclosed Please charge my:  

☐ Visa  ☐ MasterCard  ☐ American Express

Acct. No. ____________________________ Exp. Date ______

Name ________________________________

Address ______________________________

City __________________ State ______ Zip ______

Country ______________________________

*Allow 4-5 weeks for delivery.

Prices and availability subject to change without notice.
COMPUTE! Books
Ask your retailer for these COMPUTE! Books or order directly from COMPUTE!.
Call toll free (in US) 800-346-6767 (in NY 212-887-8525) or write COMPUTE! Books, P.O. Box 5038, F.D.R. Station, New York, NY 10150

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Title</th>
<th>Price*</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Machine Language for Beginners (11-6)</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Second Book of Machine Language (53-1)</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTE!s Guide to Adventure Games (67-1)</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Computing Together: A Parents &amp; Teachers Guide</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Computing with Young Children (51-5)</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPUTE!s Personal Telecomputing (47-7)</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BASIC Programs for Small Computers (38-8)</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Programmer's Reference Guide to the Color Computer</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Home Energy Applications (10-8)</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Home Computer Wars: An Insider's Account of</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Commodore and Jack Tramiel</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hardback (75-2)</td>
<td>$16.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Paperback (78-7)</td>
<td>$ 9.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Book of BASIC (61-2)</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Greatest Games: The 93 Best Computer Games of all</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Time (95-7)</td>
<td>$ 9.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Investment Management with Your Personal Computer</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>40 Great Flight Simulator Adventures (022)</td>
<td>$ 9.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>40 More Great Flight Simulator Adventures (043-2)</td>
<td>$ 9.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100 Programs for Business and Professional Use (017-3)</td>
<td>$24.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>From BASIC to C (026)</td>
<td>$16.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Turbo Pascal Handbook (037)</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Electronic Computer Projects (052-1)</td>
<td>$ 9.95</td>
<td></td>
</tr>
</tbody>
</table>

* Add $2.00 per book for shipping and handling. Outside US add $5.00 air mail or $2.00 surface mail.
NC residents add 4.5% sales tax.

Shipping & handling: $2.00/book

All orders must be prepaid (check, charge, or money order).
All payments must be in US funds.
☐ Payment enclosed.
Charge ☐ Visa ☐ MasterCard ☐ American Express

Acct. No. ___________________________________________ Exp. Date ________ (Required)
Name _____________________________________________
Address __________________________________________
City _____________________________________________ State ________ Zip ________

* Allow 4-5 weeks for delivery.
Prices and availability subject to change.
Current catalog available upon request.
\[ ZDAD = \frac{8192}{3328} \cdot \frac{160}{13} = 11695 \]
Thousands of people have already made SpeedScript COMPUTE! Publications' most popular program ever. Offering nearly every feature and convenience you expect to find in a quality word processor, SpeedScript is the perfect writing tool. With SpeedScript, writing, editing, formatting, and printing any document—from the shortest letter to the longest novel—become easier. Anything can be changed, modified, or rewritten with just a few keystrokes on your Atari 400/800, 600XL/800XL, 1200XL, or new XE and at least 24K of memory. The mechanics of writing become less intrusive—so you can concentrate on the writing, not the process itself.

SpeedScript 3.0 is our most powerful version of this easy-to-use word processor. Commands have been added, other features enhanced, to give you the best possible writing instrument.

Here are just a few of the features of this book:

• Complete program listing for SpeedScript 3.0
• Detailed documentation that shows you how to use all of SpeedScript's commands and capabilities
• "The Machine Language Editor: MLX," an entry program which insures that you'll type in SpeedScript right the first time
• SpeedScript 3.0's source code (by studying it, you'll see how SpeedScript was written)

SpeedScript is a most impressive word processor. It's a writer's tool that you can use from the moment you run it. With SpeedScript: The Word Processor for the Atari, you have a complete package—the word processor and the complete documentation.