Making Conversation: The VS-100

by Gary A. Ludwick

edited by Ryan Davis-Wright

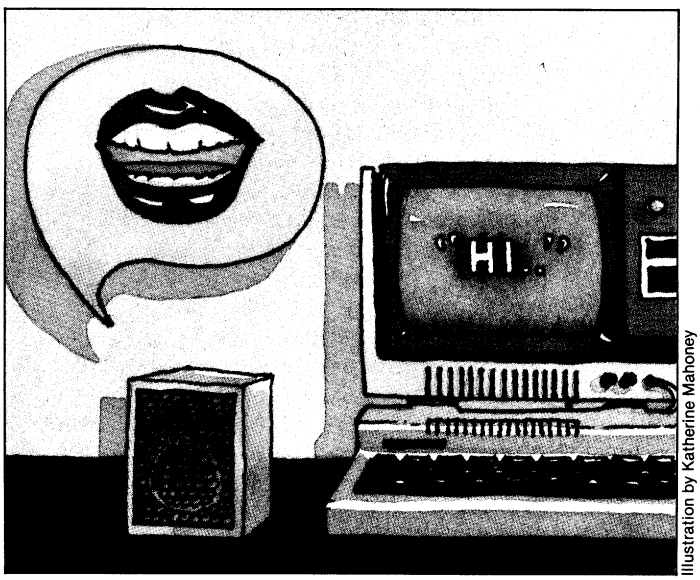

Illustration by Katherine Mahoney

If you're an avid reader of *80 Micro*, chances are you've seen an ad for a $69.95 voice synthesizer from Alpha Products. You might have wondered just how good it could be for such a low price. Well, the VS-100 does everything that Alpha claims it does, and does it surprisingly well. While it has some shortcomings, the VS-100 does an admirable job of synthesizing human speech.

The VS-100 is a compact 3- by 5-inch circuit card that plugs into either the Model I expansion port or the Model III's 50-pin I/O bus. To install it, you simply plug it in and attach the volume control knob. Power is supplied by a regulated transformer wall plug that goes to the nearest 110 volt outlet.

The only other step is to connect a speaker. Since the VS-100 contains its own amplifier, any small speaker will do. But for only $5.95, Alpha Products will supply you with one. This well-designed little speaker comes with the correct miniplug for the circuit board and it's all you really need.

The VS-100 is built around the Votrax SC-01 phoneme synthesizer. A phoneme is a basic unit of speech, a sound like "th" or "ch" or "ee." The English language has about 64 such units and the Votrax can speak them all.

Fortunately, you don't have to know anything about phonemes or their computer codes to make the VS-100 work. The Talker 2.0 software takes care of that.

The real impact of the VS-100 comes with its pairing to the Talker 2.0. While the hardware is affordable by almost everbody, the software makes it usable for even a beginner.

Before you start to make it speak, you need to understand that the Talker is designed primarily to work with Basic programs. It works under all DOSes and some machine-language programs, but if you're looking forward to a talking word processor, you might be disappointed.

How It Works

Talker comes with a wide range of choices and parameters for setting up the VS-100 (see Table 1 for a command synopsis). You can command Talker to automatically speak each word entered from the keyboard (keyboard echo) or each word that appears on the video screen (video echo). It can pronounce each letter or punctuation mark, and you can select the pitch of the voice and the speed of the delivery.

Also, you can set aside a speech buffer that will feed the phoneme codes to the VS-100 as fast as it can receive them without slowing down the computer functions. Up to 10K is reservable for this purpose, which works out to about 20 minutes of speech.

All of the control codes are sent to Talker when it is loaded from DOS Ready. For instance, "TALKER15/VW//BY//1" would load Talker into highest available memory, set aside a 4K buffer (15 x 256 bytes), suppress duplicate blanks to speed up speech delivery, and set the pitch to 1, a male voice. Every control code can also be embedded in your Basic program to change parameters or inflection whenever desired.

After you've loaded Talker, set the initial parameters, and entered Basic, you're now ready to make it speak. If you're writing a program, and the keyboard or video echo option is on, every word that you type will be spoken. Or, loading an existing program into Basic (with the video echo on) makes every word that appears on screen spoken automatically. This is a mixed blessing, however.

The main problem, of course, is that most people read faster than the VS-100 can talk. And with the speech buffer, the computer will keep on going while the VS-100 lags a screen or two behind. This can drive you crazy in short order.

If you want to cancel this, pressing the shift and space keys together will clear the buffer and bring the VS-100 to where you are currently on screen.

After five minutes trying to automatically use speech in a text adventure

game, I realized that the constant electronic chatter of the video echo option might be the easiest, but not the most satisfying, way to make the VS-100 work.

Converting an Existing Program

One other feature of Talker implements two Basic commands: Print* and Print!. Change any Basic print statement to Print* and the line within quotes that follows will be spoken, but not printed on the screen. Use Print! and the line will be spoken and printed.

The secret to effective computer-generated speech lies in the judicious use of these two statements. You have to know when to make the computer talk and when to let the user read the screen. It takes some trial and error to achieve the best mix of talking and silence, but you will quickly learn that computers (like humans) are more interesting when they occasionally keep quiet.

For those with a good word processor, the task is even simpler. Just load in your Basic program as a text file and use a global Find and Replace command to locate each print statement in your program. If it looks like something you want spoken, hit the replace key to change the statement to Print* or Print!.

Writing Your Own Talking Program

In addition, the Talker offers a convenient editor and speech/text generator. It allows you to create your own spoken phrases as subroutines.

Figure 1 is what you see on the VSEDIT screen. What is at the top is not garbage, but really the phoneme construction of the phrase, "A big hello to *80 Micro* readers." At the bottom of the screen are the commands that allow you to manipulate and fine-tune the phrase. Each of the periods represents an inflection or pitch code, and each of the PA groupings represents pauses of various lengths.

Because the VS-100 is not 100 percent perfect in its pronunciation, you will always have some tweaking to do in your sentence construction. For example, it can't distinguish between read (present tense) and read (past tense). To pronounce it correctly in the past tense, it must be typed in phonetically: "I have red this book." Reading would have to be spelled reeding, and so on. You must be conscious of the way words are spoken, as opposed to the way they are spelled.

With VSEDIT, you can type in a phrase, listen to it spoken, and then fine-tune the pronunciation and the inflection. If you've written a question, you would want to make the inflection rise at the end of the sentence for a more realistic-sounding phrase.

When you have the text/speech just the way you want it, it can be saved to disk as a line-numbered subroutine. Then reference those lines in your program each time you want the machine to talk. When these subroutines are merged back into your finished program, they even come complete with the return statement built in.

Good but Not Perfect

Before the quibbling begins, I must note that the package of the VS-100 and Talker 2.0 is an excellent bargain.

KL	Keyboard echo by letter
KW	Keyboard echo by word
KN	Keyboard echo off
VL	Video echo by letter
VW	Video echo by word
VN	Video echo off
DP	Delivery proportional
Dx	"x" is A (fastest) to N (slowest)
SY	Spell on
SN	Spell off
PN	Punctuation ignored
PY	Punctuation said
MY	Math operands said
MN	Math operands not said
BN	All spaces are pauses
BY	Duplicate blanks ignored, single blanks shortened
/1/	Lowest pitch (male)
/2/	Regular pitch
/3/	Higher pitch
/4/	Highest pitch (female)
QY	Quiet on
QN	Quiet off

Table 1. Talker 2.0 command synopsis.

But there are some things that it just won't do. Because many machine-language programs use their own keyboard and video drivers, the echo option sometimes doesn't function. That leaves word processors like Scripsit and Lazywriter speechless.

You may also have some problems with Basic programs that use high memory for their own purposes. The manual (36 pages on the VS-100 and 12 pages on the Talker 2.0) gives you some suggestions for handling this, but doesn't go into detail.

According to Alpha Products, you first load your high-memory program into Basic, come out to DOS Ready with a CMD"S to load Talker, then go back to Basic with a Basic * command (the Basic * command returns you to your location in Basic without resetting anything). Since Talker is self-relocatable, it will take the highest memory still remaining.

```
.UH2  .UH3  .PA1  .B    .I    .G    .PA1  .H    .EH   .L    .PA1
.L    .01   .U1   .PA1  .T    .IU   .U    .PA1  .A2   .A2   .Y
.T    .Y    .PA1  .M    .AH   .E1   .PA1  .K    .R    .O1   .U1
.PA1  .R    .E    .D    .ER   .Z    .PA1  PA0
```

Arrows—move cursor
A—add phoneme
<ENTER>—say all
shift [—inflection

I—insert words
D—delete phoneme
S—say word
*—save subroutine

R—remove word
C—change phoneme
K—clear all
?—recall from DISK

Figure 1. The Talker 2.0 VSEDIT screen.

But it doesn't always work. Sometimes there is no speech (with Video Echo engaged), or only part of the screen display is spoken. And sometimes the whole program crashes.

The experts at Alpha Products could probably tell me what was wrong, but technical support is only available by mail.

Conclusions

All things considered, the VS-100 and Talker 2.0 are a major accomplishment for such a low price. No one who heard my computer talk could resist typing in their own phrases and messages. That is how we found that several popular off-color phrases and words had been bleeped out by the software designers.

While the manual is well done, I wish it had gone into more detail about integrating Talker with some high-memory programs.

The speech is usually very clear and distinct with a bit of an accent. But the clarity of speech is directly proportional to the amount of fine tuning you do within your programs. In the automatic screen or keyboard echo mode, a 10–15 percent mispronunciation rate seems to be about par.

The VS-100 and software are a fascinating system with experimental and practical applications just now being explored. While it may not be a necessity, it would be hard to find more value and fun for $106. ■

Pascal in the Fast Lane

by Alan Neibauer

The Turbo Pascal compiler is a steal at $49.95. It compiles source code extremely fast (well-deserving of its Turbo name) and has extensions and refinements to Pascal that make it a complete software development tool. Even with the refinements, the syntax adheres closely to standard Pascal, so programmers will have no trouble using it.

Unfortunately, Turbo can only be used by CP/M and MS-DOS users, not TRSDOS diehards. Formats are available for the Model 2000, Model III owners with CP/M cards installed, and Model 4 users with either Radio Shack's or Montezuma Micro's CP/M.

No one who heard my computer talk could resist typing in their own messages.

Turbo Language

Turbo supports almost all the Jensen and Wirth standards, including pointers and variant records. I've used it to run a number of programs from standard Pascal texts with no editing necessary. The only items not included in the language are:

- *Dispose.* Use Mark and Release instead.
- *Packed Variables.* Turbo uses packing automatically whenever possible. The reserved word packed will be accepted by the compiler, but with no effect.
- *Page Procedure.*
- *Procedural Parameters.* Procedures and parameters cannot be passed as parameters.
- *Get and Put.* All I/O functions can be handled with extended Read and Write procedures. According to Borland, this speeds up I/O and reduces overhead.

Turbo Pascal

★★★★★

Borland International
4113 Scotts Valley Drive
Scotts Valley, CA 95066
800-227-2400 ext. 953
408-438-8400 (technical information)
Models III and 4 with CP/M
48K RAM
One Disk Drive
$49.95

Easy to use?	★★★★☆
Good docs?	★★★★★
Bug free?	★★★★★
Does the job?	★★★★★

The extensions to standard Pascal are what make Turbo outstanding. In fact, most UCSD programs will run under Turbo with little or no editing.

A complete set of dynamic strings is available and you can assign, concatenate, and test them with the normal relational operators. You can identify a variable as of type string(n), with n being the maximum string length. String arrays, both single and multidimensional, are possible through Type or Record.

In addition, string procedures include Delete (removes a substring), Insert (adds a substring), STR (conversion from integer or real to string), and VAL (converts from string to either type integer or real).

String functions include Copy (returns a substring), CONCAT (combines strings and substrings), Length (like the Basic LEN), and POS (similar to instring).

Turbo also provides full file-handling facilities including Seek to allow random access. You can treat external devices like text files and access them through the preassigned files CON:, TRM:, KBD:, LST:, AUX:, and USR:. Outputting to the line printer, for example, is as simple as including the LST: external device name in the write procedure: Write (LST:, "Turbo Pascal").

No special files need to be assigned, and the EOF, FILEPOS, and Filesize functions are also included.

When set for your terminal, Turbo offers a number of screen function procedures, such as high and low video, GOTOXY, CLREOL, and CLRSCR. For direct memory manipulation, Move performs a mass copy of a specified number of bytes, and FILLCHAR fills any range of memory with a specific value. Random number generation is also supported through Randomize, Random, and Random(I).

For the more adventurous programmer, Turbo provides absolute address variables, Chain and Execute commands, Include files, and in-line machine code. Procedures and functions are given to directly access CP/M or MS-DOS BDOS and BIOS functions and to write your own I/O drivers.

Compiler Directives

Many of Turbo's features are controlled by compiler directives that you

can set. Each directive has a default value and can be ignored by the inexperienced programmer.

Changing a default value involves including a dollar sign and the directive somewhere in the source code. This makes it easy to change system parameters for optimized code. A summary of the directives is given in Fig. 2.

The Turbo Menu

Turbo is menu-driven and includes its own WordStar-like editor. Menu options comprise the following: edit, work file, main file, compile, run, execute, save, directory, logged disk, compiler options, and quit. Selecting the edit menu option will ask you for the name of the work file then place you in the editor. If the work file is on disk, the code will be loaded and displayed.

The work file contains the current source code. You use a main file only when working with programs using the $I compiler directive to include files. You can load both the work and main files through the menu options.

You compile the current work file with the C option. You can run the work file by either compiling it first and giving the R command, or selecting R initially. If a compiled program is not already in memory, the current work file will first be compiled, then executed.

Other programs on your disk can be run from within Turbo through the X option. Save, DIR, and Logged Disk will write your code to disk, display the disk directory, and change to another disk drive.

You use the compiler Options command to select the type of compiled code desired. Normally, code is compiled to memory and can be run only from within Turbo itself. This provides maximum efficiency while writing and debugging a program. Through the compiler options, however, you can instruct Turbo to save the compiled p-code to disk, making it an executable .COM file from the operating system, or use it to create chain files.

Unfortunately, chaining files is one of Turbo's awkward features. To chain or execute files, you must assign code, data, and stack space. If not done properly, chaining will result in an out-of-memory error message. While this is explained in the manual, the instructions are not sufficient to prevent you from trial-and-error experimentation.

The Turbo Editor

The built-in Turbo editor obeys almost all of WordStar's command keystrokes. Some minor improvements have been incorporated to increase its efficiency as a source code editor. A single word can be marked as a block (control-K, control-T) and a line restore command (control-Q, control-L) has been added to restore an edited line to its original contents as long as the cursor has not left the line. While no fixed tabs are provided, an auto-indentation feature will automatically indent the start of each line to the starting position of the line above. This makes the indentation common in Pascal programs much simpler.

Operation

The compiler is where Turbo earns its name. Over 2,000 lines of p-code can be compiled per minute when it's compiling to memory. And it only takes a few seconds longer when compiling to disk. After years of teaching UCSD Pascal, Fortran, and Cobol on a variety of machines, I was amazed at how fast it was.

When the compiler encounters an error, the error number (or full message if you loaded the message file) will appear. While the messages are clear and concise, they are also more fully explained in the manual. For the exact source line error, ESC will display it and put you in the edit mode. Correct the error and repeat the process.

Several Pascal compilers have this feature and it can be a mixed blessing. First, you have no idea how many errors there are until they are all corrected. Second, some programmers appreciate receiving a program listing with all of their error flags and warnings. The list can then be examined away from the computer.

Turbo's method would be useless if it took a long time to compile the programs. You really don't want to wait 30 minutes for it to find an error. But Turbo is almost as fast as interpretive Basic's error checking.

Also, a minor error in the declaration section may cause any number of errors later in the code. By fixing the early error, you can sometimes clear up later ones. This is true of other languages besides Pascal.

B When active, standard input and output files are :CON.
C When active, control-C and control-S are obeyed during console I/O.
I When active, all I/O operations are checked for error. When followed by a file name, instructs the compiler to include a file during compilation.
R When nonactive, array indexing is not checked.
V When active, type checking is performed.
U When nonactive, control-C will not interrupt program execution.
X When active, code generation of arrays is optimized.
A For CP/M only, when active nonrecursive code is generated.
W For CP/M only, controls level of nesting allowed on "with" statements. Default is two.
K For MS-DOS only, when active stack space is checked for adequate room.

Figure 2. Compiler directives (default settings).

Turbo Programs

Along with Turbo.COM, the distribution disk includes a number of auxiliary programs. The error message file, TURBOMSG.OVR, is pure ASCII, and can be edited or translated into other languages with the editor or any word processor. TLIST.COM, a source code listing program, will print your code with line numbers and key words underlined. Since the source files are also ASCII, the system Control-P, Type or Print could also be used for printing your code.

To install Turbo for specific terminals, you use TINST.COM and TINST.DAT. A wide number of terminals are available from a TINST. You can change the editor commands with TINST, redefining the arrow or function keys for often used keystrokes.

Other files include Turbo.OVR, needed to run Pascal .COM files under CP/M-80; Error.DOK, which comprises some useful notes from the folks at Borland; and a spreadsheet program written entirely in Turbo Pascal. For a good lesson in Pascal programming techniques, examine this, run it, and then compile it. It's nothing fancy but it's a good demonstration.

The Manual

The 250-page bound manual provides a good explanation of most aspects of Turbo, except for the lack of coverage of chaining files. While it will not serve as a Pascal text editor (nor should it), the manual covers all the bases of Pascal, including more advanced topics regarding CP/M and MS-DOS implementations.

Conclusion

I've used Turbo on three different computers: a Model III, and 8- and 16-bit Zeniths. It performed flawlessly. As a programming teacher, I have to judge software critically (especially language compilers). I want my students to spend their time learning programming, not the intricacies of a particular compiler.

Turbo is easy to use and an absolute marvel at compiling code. It is a fine programming and educational tool, and a good bargain to boot. I recommend it for both the beginning and advanced Pascal programmer. ■

C Sophistication From Manx

by John B. Harrell III

I've had a lot of experience with different C compilers, but the Aztec C80 Compiler and Professional Development System is the best I've seen. It's a full implementation of C in its purest form. It faithfully supports the standards set forth for C and so produces easily portable code. If you're interested in an excellent programming language and a powerful tool for software development, the Aztec C80 system is an obvious choice.

What's Inside

The software package contains three disks and the documentation. The disks (all two-sided flippy disks) contain the compiler software and the additional features of the development system. They all come in double-density TRSDOS 6.X-format (or whatever is applicable for your system), but don't provide an operating system itself. Figure 3 lists the disk files and gives a brief description of each.

The major system components are the C80 compiler, relocatable assembler (AS), linkage editor (LN), object file librarian (LIBUTIL), DOS loader format generator (CV), and text editor (VED). The disk also contains several libraries of object modules that generate code in different ways and other features, including a powerful program text editor (Z).

Perhaps the most important part of the Aztec system is the tutorial included in the documentation—it's a must for learning the system, since the going can get a bit rough. For instance, while Aztec provides examples on how to configure your working disks, I had to spend an appreciable amount of time just building and laying them out. The software doesn't make it easy for a first-time user and I felt it was unnecessarily complicated.

Furthermore, Aztec distributes many of its standard header files in archive files you can't easily access. And you need these header files to access many of the system's functions. I spent a considerable amount of time reading the manual before I discovered the write-up on ARCV, the archive manager.

The Text Editor

Manx provides two vehicles for text editing: VED and Z. While VED is very fast, it's limited to the cursory functions of moving through the file, searching, and entering text. Z, on the other hand, provides all the features you need in a text editor.

I appreciated VED's simplicity. It provides the basic functions needed to enter a program and change it, while being easy to use. To top it off, you get VED's complete source code in an archive file; industrious programmers can easily change it to suit their tastes.

If you aren't satisfied with the limitations of VED, the Z editor should take care of your needs. This is a program text editor patterned after the Berkeley Unix editor VI. Having the capabilities of a full Unix text editor

Continued on p. 168

Files in Standard System

File Name	Description
C80/CMD	Aztec C compiler
AS/CMD	8080 relocating assembler
LN/CMD	Linkage editor
CV/CMD	TRSDOS load image generator
LIBUTIL/CMD	Object file librarian
ARCV/CMD	Source archive unpacker
VED/CMD	VED text editor
VED/ARC	VED source file archive
C/LIB	Library of non-floating point functions
M/LIB	Library of floating point functions (Model 4 only)
T/LIB	Tiny library
R/LIB, R/CMD	Fast linking library and loader
MR/LIB, MR/CMD	Fast linking library and loader (floating point)
HEADER/ARC	Source archive of header files
EXAMPL/C	Sample C program
Z/CMD	Full-screen editor (similar to Unix VI editor)
OVBGN/O, OVLOADER/O	Overlay support functions

Files in the Pro Extension

File Name	Description
LIBCSRC/ARC	Source archive for C functions
LIBASRC/ARC	Source archive for assembler functions
TRS4SRC/ARC	Source archive for Model 4 functions
TRS3SRC/ARC	Source archive for Model III functions
MATHSRC/ARC	Source archive for math functions
TINYSRC/ARC	Source archive for tiny functions
OVLY/ARC	Source archive for overlay and fast linker
HX/ARC	

Figure 3. Aztec C80 disk files.