HC Journal™
Home Computing Journal

Welcome to: Volume 3

—Intelligent Software for Personal Computers in a Home Environment

PRODUCTION NOTE: This printed journal was produced in camera-ready format with a desktop publishing system employing an Apple Macintosh and LaserWriter system, and the following software: MacPaint, MacDraw, and MacWrite from Apple Computer, Inc; PageMaker from Aldus Corporation; Microsoft Word from Microsoft; plus Desktop Magic and LaserMagic from World Class Software.

Creative Staff:
Randy Thompson
Roger Wood
Walter Hego

Editorial Consultant
Gary M. Kaplan

Notice For Atari Users:
This Volume's Feature programs rely so heavily on string arrays that we found it impractical to convert them into Atari BASIC, which does not support string arrays. So, instead of *IS-Base*, *Crossword Composer*, and *Perfect Puppy*, we designed a special Atari section. This section answers the many letters we've received requesting Atari versions of certain programs published in HCM prior to commencement of Atari coverage. We therefore present to you three of HCM's most popular number-crunching software tools: *It Figures!*, *Savings Planner*, and *Matrix Muncher*.

Home Computing Journal (HCJ) is a quarterly multi-media software subscription service containing ready-to-run productivity, education, entertainment, and utility programs on a floppy disk. The accompanying workbook contains the required support documentation plus additional technical notes and programming aids. For current single-copy and subscription pricing, please see last page. HCJ assumes no liability for errors in articles, programs, or workbook material.

Individual volumes, bulk orders, and subscriptions are available.
See last page and /or inquire for current pricing and catalog:
Home Computing Journal
P.O. Box 70248 • Eugene, OR 97401
Tel. (503) 342-4013

IS-Base

What is IS-Base? Information power—without the price.

Most database programs tend to be somewhat confusing and intimidating. Just setting up a file with the correct number and types of fields can take considerable time and pre-planning. Because *IS-Base* does not contain "fields," so to speak, it does *not* require the usual pre-organization of data files. In fact, there is very *little* that *IS-Base* has in common with other database programs at all! What then, *is IS-Base*? In a nutshell, *IS-Base* is an efficient, ridiculously-easy-to-use data storage/retrieval program that can process virtually everthing that you throw at it . . . That's *not* a small task for an information tool of this simplicity—a program that basically has only three active commands or "verbs."

How To Converse With The Program

When you run *IS-Base*, you are presented with a question mark and a blinking cursor. The question mark is your prompt, and the cursor blinks in anticipation of a command. *IS-Base* is designed around the English sentence. To converse with the program, simply enter a command and press **[RETURN]** or **[ENTER]**. If you enter a command incorrectly, a message is displayed and you are presented once again with the familiar question mark. When entering commands, you have several line editing capabilities. Refer to your computer's control capsule for the various editing features available.

The Data Disk

Before running *IS-Base*, you must have an initialized disk (a data disk) to store the information you enter. We recommend that you use a blank disk for maximum storage space. Before entering any *IS-Base* commands, your data disk should be placed into your computer's disk drive.

Different data disks can be used for different database files. Just think of each data disk as a separate file cabinet of information. To open a file cabinet, simply insert that disk into the disk drive. This can be done at any time during execution of the *IS-Base* program (except, of course, during disk access).

All entered data is stored entirely on the data disk. Because of this, you can exit the program at anytime without losing any data. Chances are, your data would survive even a power outage! During the course of the program's operation, the computer creates certain files on your data disk. The names of these files are explained in your computer's Spec Sheet elsewhere in this article. You do not have to understand, or even know about these files to operate the program. Just as long as you don't reinitailize your data disk, your data is safe.

All You Need To Know

There are really only three commands that you must learn in order to use *IS-Base*. These commands are **IS**, **FIND**, and **FORGET**. Briefly, the **IS** command allows you to enter information into the database, **FIND** allows you to search the database, and **FORGET** allows you to delete information from the database. Once you understand these three commands, you can start putting *IS-Base* to work. The following provides explanations of these commands:

Command: **IS**
Example:

```
?JOHN PARKER IS A BUSINESS ASSOCIATE
?JOHN PARKER'S ADDRESS IS 675 ALDER, SPRINGFIELD, MN 40735
?JOHN PARKER'S PHONE NUMBER IS (301) 285-2844
```

Explanation:
Here we have entered three pieces of information into the database. We have entered John Parker's address, phone number, and his relation. Now, you can see how our program achieved its name: All information is entered into the data base with a simple declarative sentence using the "IS" conjugation of the verb *to be.* So, to add anything to your database, just enter it using this simple sentence format.

Command: **FIND**
Example:

```
?FIND JOHN PARKER'S ADDRESS
Found:
JOHN PARKER'S ADDRESS IS 675 ALDER, SPRINGFIELD, MN 40735
```

Explanation:
In this example, we asked the computer to find John Parker's address. The computer responded with the statement "Found:" followed by the desired information. Use the command **FIND** * to list all the data in your database. (To understand why this works, see the section on wild cards later in this documentation.) Even though the **FIND** command is simple to use, it can contain many powerful search parameters. These parameters are also detailed later.

Command: **FORGET**
Example:

```
?FORGET JOHN PARKER'S PHONE NUMBER
Found:
JOHN PARKER'S PHONE NUMBER IS (301) 285-2844
Should I forget this? Yes
JOHN PARKER'S PHONE NUMBER IS (301) 285-2844
--> Has been forgotten
```

Explanation:
Here we have deleted John's phone number from our database. Before the computer erased the phone number, the computer asked if it was O.K. to do so. We answered yes, and so that piece of information was forgotten. In searching for the information to be forgotten, the **FORGET** command accepts the identical search parameters as the **FIND** command. See the following section for a description of the various search parameters available.

Search Parameters

Search parameters are what you use to specify information that you are searching for. You use search parameters to find information that will be viewed, forgotten, or edited. The *IS-Base* commands **FIND**, **FORGET**, and **EDIT** all use search parameters. We will use the **FIND** command for explanatory purposes, but the following examples are equally relevant for the **FORGET** and **EDIT** commands. While reading over these examples, assume that the following information has been entered into the database:

```
?JOHN PARKER IS A BUSINESS ASSOCIATE
?JOHN PARKER'S ADDRESS IS 675 ALDER, SPRINGFIELD, MN 40735
?JOHN PARKER'S PHONE NUMBER IS (301) 285-2844
?CRAIG HENDERSON IS A BUSINESS ASSOCIATE
?CRAIG HENDERSON'S ADDRESS IS UNKNOWN
?CRAIG HENDERSON'S PHONE NUMBER IS UNLISTED
?A MYSTERIOUS FELLOW IS CRAIG HENDERSON
```

Now we'll describe the 5 basic search parameter formats. The <...> symbol represents the search parameters entered by you. All commands and keywords are in bold.

Format 1: **FIND** <...> **IS** <...>
Example:

```
?FIND JOHN PARKER'S PHONE NUMBER IS (301) 285-2844
Found:
JOHN PARKER'S PHONE NUMBER IS (301) 285-2844
```

Explanation:
This is the most basic and explicit search format. Here, you specify letter for letter the piece of information that you are looking for. You use this search format when you are looking for one specific piece of information.

Format 2: **FIND** <...>
Example:

```
?FIND JOHN PARKER'S ADDRESS
Found:
JOHN PARKER'S ADDRESS IS 675 ALDER, SPRINGFIELD, MN 40735
```

Explanation:
In this format, search parameters are provided for only the left side of the **IS** command. Any piece of information entered as "JOHN PARKER'S ADDRESS **IS** <...>" will be found. It does *not* matter what was entered after the **IS** command.

Format 3: **FIND WHO IS** <...> or **FIND WHAT IS** <...>
Example:

```
?FIND WHO IS A BUSINESS ASSOCIATE
Found:
JOHN PARKER IS A BUSINESS ASSOCIATE
CRAIG HENDERSON IS A BUSINESS ASSOCIATE
```

Explanation:
In this format, search parameters are provided for only the right side of the **IS** command. Any piece of information entered as "<...> **IS** A BUSINESS ASSOCIATE" will be found. It does *not* matter what was entered before the **IS** command. The keywords **WHO** and **WHAT** are interchangable.

IS-Base Spec Sheet: Apple II

Control Capsule

Key	Function
Cursor left	Move left ←
Cursor right	Move right
Delete	Delete a character
CTRL I	Insert a space
CTRL E	Erase current entry
Return	Accept entry
CTRL S	Freeze screen/printer output
ESC	Escape current operation

Disk Specs

When you first boot *IS-Base*, the location of your data disk defaults to slot 6, drive 1. You can change this from drive 1 to drive 2 with the **DRIVE** command. Entering the command **DRIVE 2**, directs *IS-Base* to use drive 2 as its default drive. To change the default back to drive 1, simply enter the command **DRIVE 1**.

The following is a list of the files that are created on your data disk by *IS-Base*.

Filename	Function
ISBASE.DAT	Holds all entered data
ISBASE.KEY	Holds the function key definitions
ISBASE.TMP	A temporary data file

Note: If using an Apple IIc, IIe, or IIgs do *not* run *IS-Base* with with 80-column mode activated—strange things may occur when accessing the printer.

Function Keys

Because the Apple computer has no designated function keys, we had to ad lib. Function keys 1 through 10 are achieved by holding down the [OPEN APPLE] key in conjunction with a number key 1 through 0. So, function key 1 is [OPEN APPLE] 1 while function key 10 is [OPEN APPLE] 0. Only function keys 1 through 9 are re-definable with the **KEY** command. Function key 10 always produces that last command that you entered. The default function key definitions are as follows:

Function Key	Definition
[OPEN APPLE] 1	FIND
[OPEN APPLE] 2	FIND *
[OPEN APPLE] 3	FORGET
[OPEN APPLE] 4	EDIT
[OPEN APPLE] 5	PRINTER
[OPEN APPLE] 6	SORT
[OPEN APPLE] 7	CRUNCH
[OPEN APPLE] 8	CLS
[OPEN APPLE] 9	BYE

Note: Because the OPEN APPLE key is only available on Apple IIe, Apple IIc, and Apple IIgs, other models cannot use the function key option.

Format 4: **FIND ALL** <...>
Example:

```
?FIND ALL CRAIG HENDERSON
Found:
CRAIG HENDERSON IS A BUSINESS ASSOCIATE
A MYSTERIOUS FELLOW IS CRAIG HENDERSON
```

Explanation:
In this format, search parameters are provided for either the left or right side of the **IS** command. Any piece of information entered as "CRAIG HENDERSON **IS** <...>" or "<...> **IS** CRAIG HENDERSON" will be found.

Format 5: **FIND ALL RELATED TO** <...>
Example:

```
?FIND ALL RELATED TO A BUSINESS ASSOCIATE
Found:
JOHN PARKER IS A BUSINESS ASSOCIATE
JOHN PARKER'S ADDRESS IS 675 ALDER, SPRINGFIELD, MN 40735
JOHN PARKER'S PHONE NUMBER IS (301) 285-2844
CRAIG HENDERSON IS A BUSINESS ASSOCIATE
CRAIG HENDERSON'S ADDRESS IS UNKNOWN
CRAIG HENDERSON'S PHONE NUMBER IS UNLISTED
A MYSTERIOUS FELLOW IS CRAIG HENDERSON
```

IS-Base Spec Sheet: C-64

Control Capsule

Key	Function
Cursor left	Move left
Cursor right	Move right
DEL	Delete a character
INST	Insert a space
HOME	Move cursor to beginning of line
CLR	Erase current entry
RETURN	Accept entry
Commodore key	Freeze screen/printer output (keep held down)
Left-arrow	Escape current operation

Function Keys

The **KEY** command is used to re-define the Commodore function keys located on the right of the keyboard. Only function keys 1 through 7 are re-definable with the **KEY** command. Function key 8 always produces that last command that you entered. The default function key definitions are as follows:

Function Key	Definition
f1	FIND
f2	FIND*
f3	FORGET
f4	EDIT
f5	PRINTER
f6	SORT
f7	BYE

Disk Specs

When you first boot *IS-Base*, the location of your data disk defaults to device 8 (drive 8). You can change this from drive 8 to drive 9 with the **DRIVE** command. Entering the command **DRIVE 9**, directs *IS-Base* to use drive 9 as its default drive. To change the default back to drive 8, simply enter the command **DRIVE 8**.

The following is a list of the files that are created on your data disk by *IS-Base*.

Filename	Function
IS-BASE.DAT	Holds all entered data
IS-BASE.KEY	Holds the function key definitions
IS-BASE.TMP	A temporary data file

Compiled BASIC

There are two versions of *IS-Base* on your HCJ Volume 3 disk. These versions are IS-BASE.BAS and IS-BASE.COM. The file IS-BASE.BAS is a Commodore BASIC 2.0 version of *IS-Base*. It is supplied on your disk so that you can see how the program works. We do *not* recommend that you run this version, as it is very slow. The file IS-BASE.COM is a compiled version of IS-BASE.BAS. This version can be **LOAD**ed, **SAVE**d, and **RUN** like any BASIC program. We recommend that you run the compiled version every time you use the program, as it operates much faster than its BASIC counterpart. We compiled *IS-Base* using the Abacus Software BASIC 64 compiler.

Explanation:
This format is a variation on the **FIND ALL** format. Search parameters are provided for either the left or right side of the **IS** command. Any information entered as "BUSINESS ASSOCIATE IS <...>" or "<...> IS BUSINESS ASSOCIATE" will be found.

Once a record is found using the **FIND ALL** format, however, the search goes one level deeper. You see, when the computer finds the line "JOHN PARKER IS A BUSINESS ASSOCIATE" it stops its search for "A BUSINESS ASSOCIATE" and goes looking for the name "JOHN PARKER" anywhere within the data file (left and right of the **IS** command, and even inside a statement like "*JOHN PARKER*'S ADDRESS IS"). When it finds "JOHN PARKER," that piece of information is output to the screen. When no more "JOHN PARKERs" can be found, the search resumes again for "A BUSINESS ASSOCIATE." This process is repeated when "CRAIG HENDERSON IS A BUSINESS ASSOCIATE" is found.

The All-Important *Wild Card*

Wild cards are one of the search parameter's most important features. If you have used DOS on an IBM PC or compatible, then you probably know what wild cards are and how much they can help. A wild card is a character that can represent one, none, or many characters. *IS-Base* uses the asterisk (*) character as a wild card. Because of this, you should *not* use asterisks when entering a piece of information with the **IS** command.

Let's say that you want to find all the information that you have on John Parker. You could enter the commands

```
?FIND JOHN PARKER
?FIND JOHN PARKER'S ADDRESS
?FIND JOHN PARKER'S PHONE NUMBER
```

and get all the information you have on him. But having to enter all three of these commands, would quickly become tiresome. The problem worsens when you start entering more information on John, such as business phone or date of birth. With wild cards, however, you can get *all* of John Parker's information with just one command:

```
?FIND *JOHN PARKER*
```

Here, no matter what appears before or after the name "JOHN PARKER," we find the proper information.

Wild cards are especially helpful when you can't remember exactly how you entered something into the database. By using wild cards, you can enter what you *do* remember, add a few wild cards in with your search parameters, and chances are that the computer will find the correct information. Use the command **FIND *** to find *all* information contained in the database.

Here are some examples on wild card matching: *DE matches ABCDE, JADE, and DE, but not DEAF or DEL. The string AB* matches ABCDE, ABSOLUTE, and AB, but not SCAB or TAB. The string *AN* matches ANT, FAN, AN and CANTELOPE. Finally, W*E matches WHALE, WHITE, and WE, but not WET or ANSWER.

Wild cards can be used *anywhere* within your search parameters, no matter which search parameter format you use. As you can see, wild cards add great flexibility to your *IS-Base* information retrieval.

All The Extras

Now that you've learned the "All You Need To Know" commands, and you've conquered search parameters, you're ready to learn the rest of *IS-Base's* commands. The following is a detailed explanation of *IS-Base's* extra commands. Once again, let's assume that the following has already been entered into the database using the **IS** command:

```
?JOHN PARKER IS A BUSINESS ASSOCIATE
?JOHN PARKER'S ADDRESS IS 675 ALDER, SPRINGFIELD,
MN 40735
?JOHN PARKER'S PHONE NUMBER IS (301) 285-2844
?CRAIG HENDERSON IS A BUSINESS ASSOCIATE
?CRAIG HENDERSON'S ADDRESS IS UNKNOWN
?CRAIG HENDERSON'S PHONE NUMBER IS UNLISTED
?A MYSTERIOUS FELLOW IS CRAIG HENDERSON
```

The <...> symbol represents information entered by you. All commands and keywords are in bold.

Command:
EDIT <...> **IS** <...>
or
EDIT <...>
or
EDIT WHAT IS <...> or **EDIT WHO IS** <...>
or
EDIT ALL<...>
or
EDIT ALL RELATED TO <...>

Example:

```
?EDIT JOHN PARKER IS A BUSINESS ASSOCIATE
 JOHN PARKER IS A BUSINESS ASSOCIATE
>JOHN PARKER IS A NEIGHBOR
```

Explanation:
This command allows you to edit information found in the database. Once a piece of information that matches your search parameters is found, it is printed to the screen and you are given the chance to change the information. In the example above, we have changed John Parker from a business associate to a neighbor. To

differentiate between the normal command mode and edit mode, you are given a greater-than sign (>) as an input prompt instead of a question mark (?). The original line is printed above the edit line for your reference. This way, it is easy for you to identify the changes you've made before pressing [RETURN] or [ENTER].

If you press [ESC] (back-arrow on the C-64), edit mode is aborted without any changes made to the information found. To abort edit mode on the TI-99/4A, just enter a blank line. If the **EDIT** command finds several matches to your search parameters, each piece of information found is brought up, one by one, for you to edit.

Command: **CHECK ON** and **CHECK OFF**
Example:

```
?CHECK ON
Definition check is on
?CRAIG HENDERSON'S PHONE NUMBER IS (221) 993-3772
Should I forget that-->CRAIG HENDERSON'S PHONE NUMBER
IS UNLISTED? Yes
CRAIG HENDERSON'S PHONE NUMBER IS UNLISTED --> Has been
forgotten
?CHECK OFF
Definition check is off
?CRAIG HENDERSON'S PHONE NUMBER IS NOW AVAILABLE
```

Explanation:
These two commands allow you to decide whether to forget old information because new, similar information is being added. When **CHECK** is **ON**, then newly entered information is checked against previously entered information for similarities. If a similarity is found, then you are asked if you wish to forget the old information. You may answer yes or no. To be considered similar, the information found on the left sides or right sides of the **IS** command must match perfectly on both the new and old information. In the example above, the phrase "CRAIG HENDERSON'S PHONE NUMBER" is what makes the two pieces of information similar. **CHECK OFF** turns this feature off.

Command: **KEY<...>**
Example:

```
?KEY1 IS A BUSINESS ASSOCIATE
?KEY2 FIND *
```

Explanation:
This command allows you to assign a group of characters to your function keys. The command **KEY1** defines function key 1, while **KEY7** defines function key 7. Whenever a function key is pressed, the characters assigned to that key are output. This can save much typing time when you are keying in several similiar types of information. The **KEY** command all by itself resets all function keys to their default definitions. For a list of the available function keys and their default definitions, see your computer's Spec Sheet.

Command: **SAVE KEYS** and **LOAD KEYS**
Example:

```
?KEY1 IS A BUSINESS ASSOCIATE
?KEY2 FIND *
?SAVE KEYS
```

Explanation:
These two commands allow you to save and load the current function key definitions to disk.

Command: **HELP**
Example:

```
?HELP
F1 - IS A BUSINESS ASSOCIATE
F2 - FIND *
F3 - FORGET
F4 - EDIT
F5 - PRINTER
F6 - CRUNCH
F7 - SORT
F8 - HELP
F9 - BYE
```

Explanation:
This command lists the current function key definitions.

Command: **PRINTER ON** and **PRINTER OFF**
Example:

```
?PRINTER ON
?FIND *
?PRINTER OFF
```

Explanation:
These two commands inform the program whether data should be sent to the printer or not. When the **PRINTER** is **ON** all information output to the screen is also output to the printer.

Command: **CLS**
Example:

```
?CLS
```

Explanation:
This command clears the screen. If **PRINTER ON** is activated, then a form feed is sent to the printer.

Command: **CRUNCH**
Example:

```
?CRUNCH
```

IS-Base Spec Sheet: IBM-PC/PCjr

Control Capsule

Key	Function
Cursor left	Move left
Cursor right	Move right
Backspace	Erase character left of cursor
Delete	Delete a character
Insert	Toggle insert mode
Home	Move cursor to beginning of line
End	Move to the end of line
Ctrl Backspace	Erase current entry
Enter	Accept entry
Ctrl Num Lock (PC)	Freeze screen/printer output
Fn Pause (PCjr)	Freeze screen/printer output
HOLD (Tandy)	Freeze screen/printer output
Esc	Escape current operation

Disk Specs

When you first boot *IS-Base*, the location of your data disk defaults to the active drive (usually drive A). You can change the default drive with the **A:** and **B:** commands. Entering the command **B:** directs *IS-Base* to use drive B as its default drive. To change the default back to drive A, simply enter the command **A:**.

The following is a list of the files that are created on your data disk by *IS-Base*.

Filename	Function
IS-BASE.DAT	Holds all entered data
IS-BASE.KEY	Holds the function key definitions
IS-BASE.TMP	A temporary data file

Function Keys

The **KEY** command is used to re-define the PC's function keys. Only function keys 1 through 9 are re-definable with the **KEY** command. Function key 10 always produces that last command that you entered. The default function key definitions are as follows:

Function Key	Definition
F1	FIND
F2	FIND *
F3	FORGET
F4	EDIT
F5	PRINTER
F6	SORT
F7	CRUNCH
F8	CLS
F9	BYE

Turbo Charged

The IBM version of *IS-Base* was written in Turbo Pascal, available from Borland International. Turbo Pascal compiles into .COM files that are executable from DOS. Because Turbo Pascal is a compiled language, it produces very fast and effecient programs. To run the IBM Version of *IS-Base* you may use the *HCJ Director* program or boot your system, insert your HCJ Volume 3 disk into the active drive, and enter IS-BASE at the DOS prompt.

IS-Base Spec Sheet: TI-99/4A

Control Capsule

Key	Function
FCTN S	Move left
FCTN D	Move right
FCTN 1	Delete a character
FCTN 2	Activate insert mode
FCTN 3	Erase current entry
ENTER	Accept entry
SPACE	Freeze screen/printer output (hold down)
FCTN 9	Escape current operation

Function Keys

Because the TI-99/4A does not allow programs to mask for different keypresses during an **ACCEPT AT** command, we were *not* able to add the function key option in the TI-99/4A version of *IS-Base*. Without function keys, the TI-99/4A version does not support the following *IS-Base* commands: **KEY, HELP, SAVE KEYS,** and **LOAD KEYS**.

Disk Specs

When you first boot *IS-Base*, the location of your data disk defaults to drive 1 (DSK1.). With the commands **DSK1.** and **DSK2.** you can change this from drive 1 to drive 2. Entering the command **DSK2.** directs *IS-Base* to use drive 2 as its default drive. Simply enter the command **DSK1.** to change the default back to drive 1.

The following is a list the files that are created on your data disk by *IS-Base*.

Filename	Function
IS-BASE_DT	Holds all entered data
IS-BASE_TP	A temporary data file

Printer Parameters

The default printer parameters for the TI-99/4A version of *IS-Base* are "RS232.BA=9600.DA=8". These parameters take effect whenever you use the *IS-Base* command **PRINTER ON**.

There are two ways in which you can change the default printer parameters. The first option is to use an *IS-Base* command that is unique to the TI-99/4A computer. This command is described below:

Command: **PR=<...>**
Example:

```
?PR=PIO
```

Explanation:
This command allows you to change the printer parameters.

The second option for changing the default printer parameters is to edit line 220 of the program file IS-BASE so that it sets the variable **PR$** to the parameters of your choice.

Note: This program requires Extended BASIC, but *not* the 32K Memory Expansion. If you own the 32K Memory Exansion, however, we suggest that you use it.

Explanation:
If you have made several deletions to the your data file, it is possible to shrink the data file so that it takes up less room on the disk and is searched more quickly. The **CRUNCH** command goes through and compacts your data file. This option is not available on the Commodore 64 because the data file is always kept "crunched" in the C-64 version of *IS-Base*.

Command: **SORT** or **SORT LEFT** or **SORT RIGHT**
Example:

```
?SORT
```

Explanation:
Data is saved and listed in the order in which it is entered. The **SORT** command allows you to change this order. A **SORT** or **SORT LEFT** alphabetically sorts your file according to the left side of each statement (everything left of the **IS** command). A **SORT RIGHT** alphabetically sorts your file according to the right side of each statement (everything right of the **IS** command).

Command: **BYE**
Example:

```
?BYE
```

Explanation:
This exits the *IS-Base* program and returns you to the standard power-on state of the computer. Because your data is always stored to disk, there is no need to worry about lost data when leaving the *IS-Base* program.

Sample Databases On Disk

Because of its flexibility, *IS-Base* can keep track of any type of information that you wish to store: addresses, magazine indexes, record collections, trivia questions, inventories, etc. To get you started, we have provided two sample files on your HCJ Volume 3 disk. In the file CAPITALS, we have entered the capitals of the United States. In the file MADONNA, we have entered information (song titles, writers, and musicians) about the musical pop star Madonna's first three record albums.

To use either of these files, you must first copy the data file (CAPITALS or MADONNA) to an empty data disk using the file name that your version of *IS-Base* recognizes as its data file. On the Apple, this means copying the data file of your choice (using the Filer program provided on your HCJ Volume 3 disk) to the new data disk with ISBASE.DAT as the file name. On the C-64, copy the data file (using one of the file copier programs supplied with your disk drive) to the new data disk with IS-BASE.DAT as the file name. On IBM or compatibles, copy the data file (using DOS) to the new data disk with IS-BASE.DAT as the file name. On the TI-99/4A, copy the data file (using the Disk Manager Cartridge) to the new data disk with IS-BASE_DT as the file name. These file names are described in the Spec Sheet for your computer.

The CAPITALS file shows off *IS-Base's* educational possibilities. With all of the states' capitals at your fingertips, you can use *IS-Base* to quiz yourself or a friend on geographical knowledge. The capitals were entered in the following format:

```
?THE CAPITAL OF ALASKA IS JUNEAU
?THE CAPITAL OF CALIFORNIA IS SACRAMENTO
etc.
```

So, to find the capital of South Dakota, for example, simply enter the following command:

```
?FIND THE CAPITAL OF SOUTH DAKOTA
Found:
THE CAPITAL OF SOUTH DAKOTA IS PIERRE
```

To find the state that a particular capital city is located in, enter this:

```
?FIND WHAT IS TALAHASSEE
Found:
THE CAPITAL OF FLORIDA IS TALAHASSEE
```

The MADONNA file emphasizes *IS-Base's* indexing and searching capabilities. This file identifies all songs on the three albums by album and by songwriter(s). For example, it's easy to find out both the songwriter and which album contains the song *La Isla Bonita* by using the **FIND** command:

```
?FIND LA ISLA BONITA
Found:
LA ISLA BONITA IS ON THE ALBUM TRUE BLUE
LA ISLA BONITA IS WRITTEN BY MADONNA, PATRICK LEONARD,
AND BRUCE GAITCH
```

As this example shows, we've entered all song titles on the left side of the **IS** command. Because of this, we don't need to use the **FIND ALL** format to locate information about any of the songs. Now, let's say we want to find out more about a particular songwriter. If we just enter **FIND ALL** PATRICK LEONARD, here is the result:

```
?FIND ALL PATRICK LEONARD
Found:
PATRICK LEONARD IS A PRODUCER OF THE ALBUM TRUE BLUE
PATRICK LEONARD IS DOING DRUM MACHINE PROGRAMMING ON
THE ALBUM TRUE BLUE.
```

Using wild cards (*), however, will give you even more information. Try entering the following command on your own computer to discover Patrick Leonard's other songs:

```
?FIND ALL *PATRICK LEONARD*
```

Be sure the MADONNA file has been properly renamed for your computer system, as explained above.

Crossword Composer

You think up the words, the computer interleaves them into a puzzle.

If you've ever tried to create a crossword puzzle, then you know what frustration is. Attempting to fit a slew of words into one cohesive group is no small task. Such a procedure is time consuming and requires a lot of patience. This type of trial-and-error letter matching, however, is perfect for a computer. Computers never get bored; they think quickly; and they just "love" repetitive operations. The program *Crossword Composer* really gives your computer a chance to show off these talents. With *Crossword Composer*, you enter up to 100 words and then let the *computer* fit the words together.

Crossword Composer operates from one main screen (see Figure 1). The Puzzle Box appears on the left. This box contains your crossword puzzle. The Puzzle Box is 20 characters wide and 20 characters tall. When you first run the program, this box is empty. Directly above the Puzzle Box is the title of the puzzle. The title is determined by the file name used when you save your puzzle. The puzzle's title remains "UNTITLED" until you save or load a puzzle.

On the right side of the screen is the program's menu. A selected menu option appears highlighted. To select an option, use your computer's cursor-up and cursor-down keys. To activate the selected option, press **[RETURN]** (**[ENTER]** on the IBM and TI-99/4A). The menu options are:

Edit Words	Save Puzzle
Make Puzzle	Load Puzzle
Erase Puzzle	Future Plans
Print Puzzle	Exit Program

Edit Words

Before you can create a puzzle, you must enter the words and clues that will make up the puzzle. Words and clues are entered and edited by choosing the Edit Words option.

When you select this option, you are brought to the Edit Words' screen (see Figure 2). This screen shows five boxes at a time (4 on the TI-99/4A). These boxes are where you enter your words. Each box is numbered (0 through 99), informing you which of the 100 words you are currently editing. The screen line below each box is where you enter the word's clue. Using your cursor keys, you can move from word to clue, clue to word, etc.

When you attempt to move the cursor off the screen, the screen updates to display the next or previous words in the list (depending on whether you were moving up or down). By using certain keypresses, you can jump through the word list a screen at a time, or just a word at a time. There are also keypresses for cutting and pasting words to and from the word list. For a complete list of Edit Word's editing functions, refer to your computer's Control Capsule.

Words cannot contain spaces—any spaces found in a word are taken out before the word is placed into the puzzle. Also, words must be two or more letters in length. Words do not have to have clues, but crossword puzzles without clues are not much fun!

The Order Of Words

The order in which you enter words can determine the final look of your puzzle. When the computer assembles your crossword, it places one word at a time into the puzzle, moving sequentially through the word list. For best results, therefore, you should put your longest words at the top of the list. This way, the computer will have an easier time placing the words found at the bottom of the list. For the same reason, it's a good idea to include words with commonly used letters at the top of the list. The editor's cut-and-paste functions aid considerably in the reordering of words. The order of words is not entirely critical, however, because the computer repeatedly cycles through the word list, placing any words that could not fit in the first pass. Just remember: a little forethought can help in producing a more complete puzzle.

Pressing [ESC] (back-arrow key on the C-64 or [FCTN] 9 on the TI-99/4A) exits the word editor and returns you to the main menu screen.

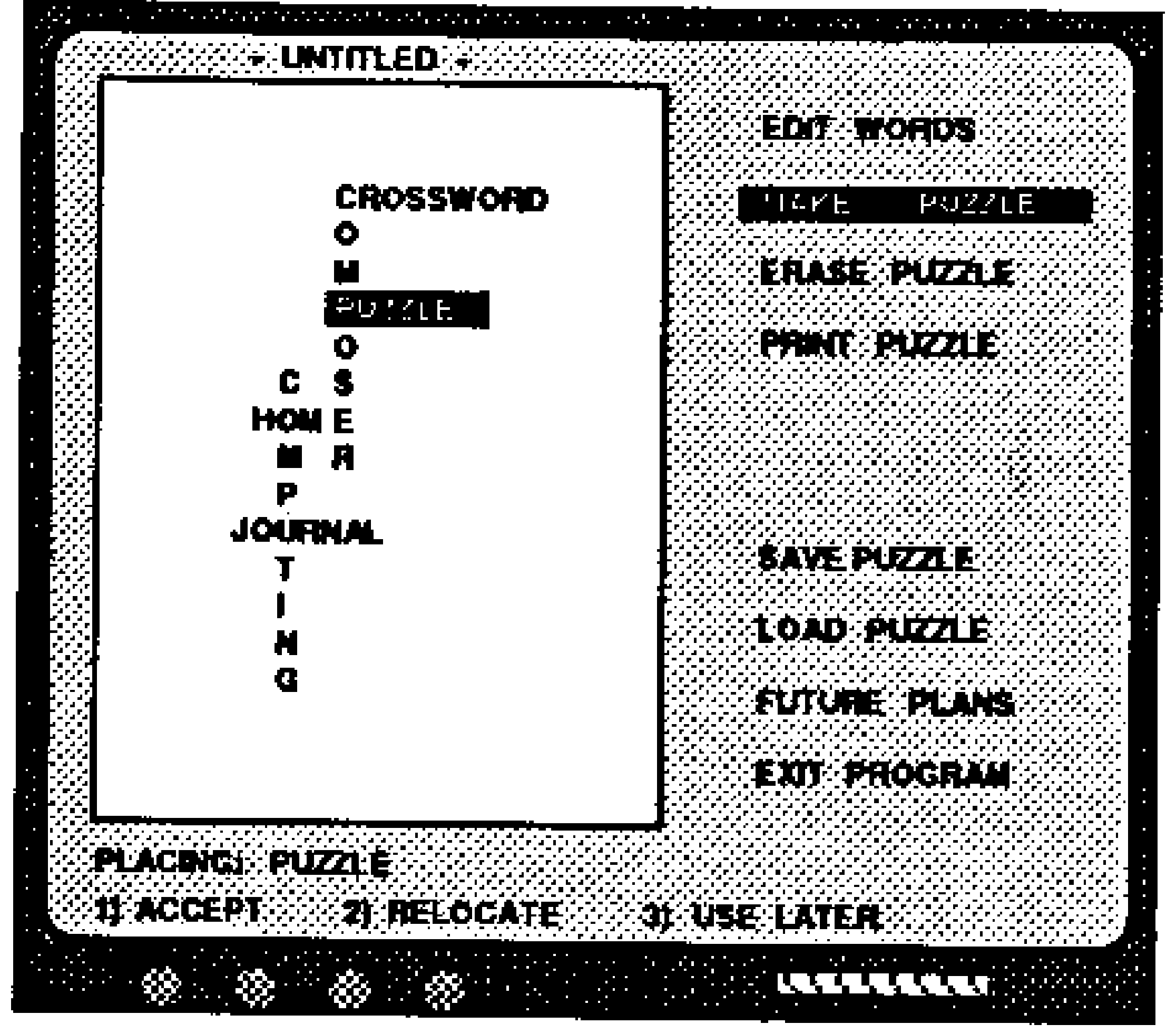
Make Puzzle

This option allows you to make a crossword puzzle using the words currently in the word list. While making a puzzle, the screen line directly below the puzzle box shows which word is currently being placed, while the line below that displays your options, including the keypresses you use to access these options.

The first step to making a puzzle is to place the first word inside the Puzzle Box. Your options for placing the first word are: (1) accept the current position of the word, (2) toggle the word between being displayed across or down, or (3) move the word within the Puzzle Box using the cursor keys. Because this is the first word in the Puzzle Box, it can be placed anywhere within the Puzzle Box's boundaries.

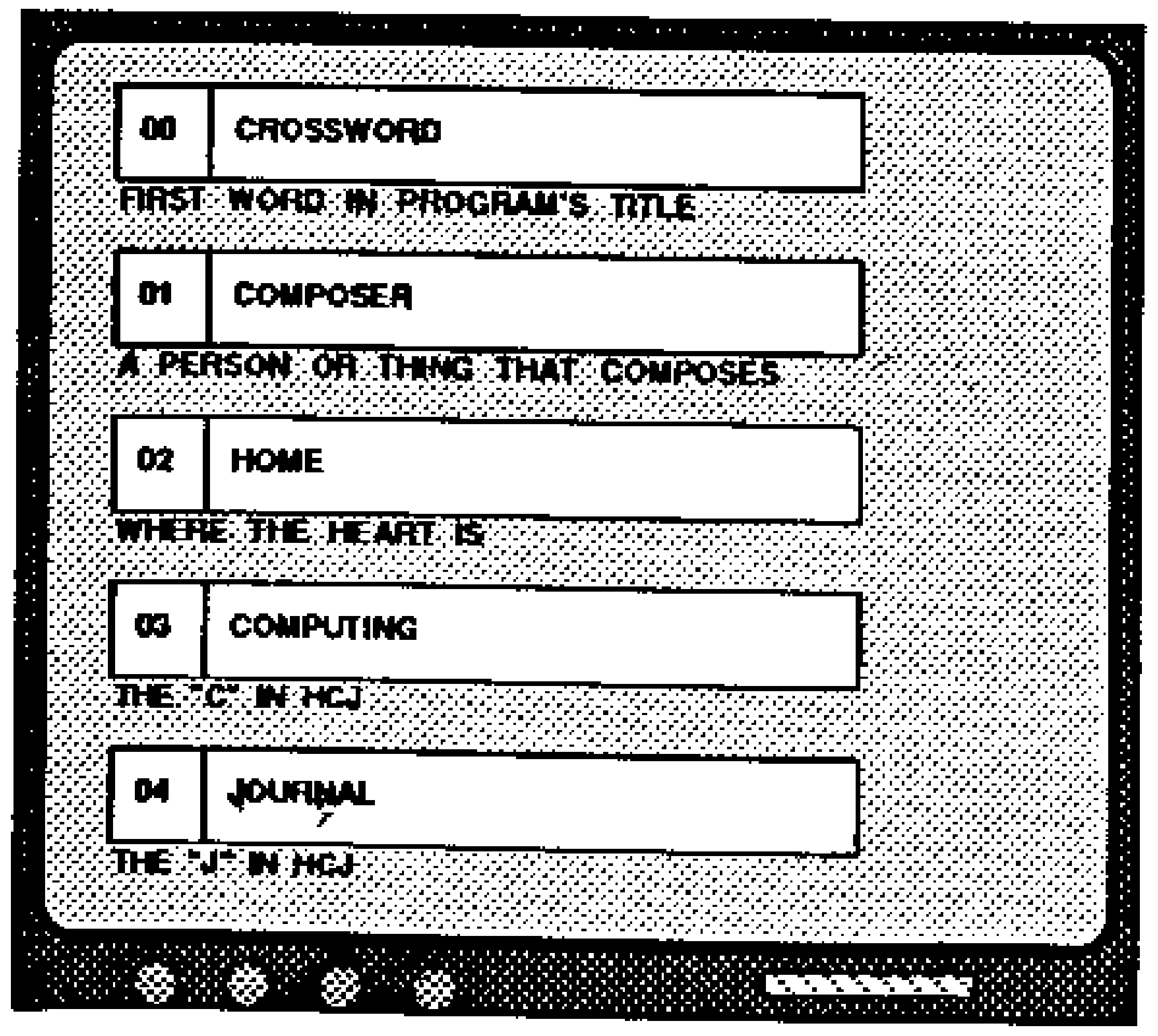
Once the first word is placed, all following words offer these options: (1) accept the current position of the word, (2) relocate the word in the puzzle, or (3) use the word at a later time. When a word is placed into the puzzle, the word appears in the Puzzle Box in reverse video. As described previously, you can accept the current location of the word, or relocate the word. Using the Relocate option, you can cycle through all of the word's possible positions; this may be several positions, or just one. The Use Later option comes in handy when you do

Figure 1.
This is Crossword Composer's main screen. The program's menu appears on the right.



PLACING: PUZZLE
1) ACCEPT 2) RELOCATE 3) USE LATER

Figure 2.
This is the Edit Word's screen. The puzzle's words appear in boxes with their clues below.



00 CROSSWORD
FIRST WORD IN PROGRAM'S TITLE

01 COMPOSER
A PERSON OR THING THAT COMPOSES

02 HOME
WHERE THE HEART IS

03 COMPUTING
THE "C" IN HCJ

04 JOURNAL
THE "J" IN HCJ

Crossword Composer Spec Sheet: Apple II

Control Capsule

Edit Word Mode

KEY	FUNCTION
Delete	Delete a character
CTRL I	Insert a space
RETURN	Move to clue or next word
Cursor down	Move to clue or next word
Cursor up	Move to word or previous clue
CTRL Q	Move to previous word/clue
CTRL A	Move to next word/clue
CTRL W	Move back by 5 words
CTRL S	Move forward by 5 words
CTRL X	Cut word/clue from list
CTRL V	Paste word/clue into list
ESC	Return to menu

To run the Apple version of *Crossword Composer*, simply **LOAD** and **RUN** the program file CROSSWORD. You may also use the HCJ Director program. This program should be run in 40-column mode only. If your computer supports 80-column mode, be sure it is not active when you run this program.

Printer Drivers

There are two different printer drivers on the Apple HCJ Volume 3 disk:

File name	Printer
`ALL.PRT`	Any printer—no graphics
`EPSON.PRT`	Epson-compatible dot-matrix printers

The program file CROSSWORD has the ALL.PRT driver installed. In order to use the Epson-compatible printer driver, you must install it into the CROSSWORD program. To install a driver, follow these steps:

1) **LOAD** the CROSSWORD program. For example:
```
LOAD CROSSWORD
```
2) **BLOAD** the printer driver of your choice. For example:
```
BLOAD EPSON.PRT
```
3) **SAVE** the program under the file name of your choice. For example:
```
SAVE CROSSWORD.EPSON
```

Because the HCJ disk is write-protected, you must use a backup of the disk to complete this procedure.

File Names

File names for saving and loading puzzles can be up to 11 characters in length. When saved to disk, your file will appear in the directory with a .PUZ extension. You must use prefixes to specify which disk/drive you want to access. Everytime you enter a file name, a default prefix is supplied (usually indicating the disk in slot 6, drive 1). So, if you don't mind the current prefix, you can simply enter your file name and press **[RETURN]**.

not like any of the positions where the word can currently be placed. Choosing this option simply sets the word aside, where it will be used again after the rest of the words in the list are placed. Take note: To avoid interjecting new unsolicited words into the puzzle, words are *never* placed directly next to another word.

This process repeats for every word in the list, until the puzzle is complete. If any words did not fit into the puzzle, then the computer tells you which words were not used.

If you have a puzzle already assembled in the Puzzle Box when you select the Make Puzzle option, that puzzle will be cleared in preparation for your new puzzle. When rebuilding a puzzle, however, the computer always tries to place the words in their previous positions—duplicating your original puzzle. This way, you can make modifications to the puzzle's words or format without totally disrupting its look. If you modify the puzzle's words too drastically, or simply take an important word from the list, the computer probably will not be able to re-create the format of the previous puzzle.

Editing Words After Making A Puzzle

If you have a puzzle assembled in the Puzzle Box when you enter Edit Words, you'll see asterisks to the right of some of the words in the word list. These asterisks signify that a word is currently being used in the puzzle. Words that did not fit into the puzzle do not have an asterisk. You can use this information to weed out these "outcast" words, or to relocate them in the list.

If you do not want to disrupt your current puzzle, you must be cautious of how you treat words with asterisks. If you delete, or even change the spelling of a word that is used in the puzzle, your puzzle is cleared and you are forced to rebuild it. You see, by modifying a word that is used in the puzzle, you have corrupted the position of other words. Of course, you can do whatever you want with words not marked by an asterisk.

Erase Puzzle

If you want to clear the current puzzle from memory and start from scratch, choose this option. The Erase Puzzle option clears the puzzle from the Puzzle Box, erases all words in the word list, and resets the puzzle's title to "UNTITLED." Because of the serious nature of the Erase Puzzle option, you are asked if you really want to erase the puzzle. If you stumbled into this option by mistake, simply answer NO.

Print Puzzle

Once you have made a puzzle that you are satisfied with, you will want to print your puzzle. Three pages are produced when you choose this option. The first page is the problem page, displaying the numbered boxes in which the words are entered. The second page is the clue page (clues may take up more than one page, depending on the number of words in your puzzle). The third and final page is the puzzle's solution. You may wish to keep this page when challenging someone with your puzzle.

In order to optimize this all-important print feature for your computer-printer setup, we offer two different types of printout. The default printer method uses only characters to create the puzzle—employing asterisks to draw the puzzle box. This method works with all printers including daisy wheel printers.

We also allow for a more elegant graphic output by including special "printer drivers" on each computer's disk. (A printer driver is simply the software necessary for a program to send special output to a printer.) This graphic method is more time consuming, but the result is superior. In order to get a graphic printout, your program needs to have a special printer driver installed in it. Also, due to the printer-specific nature of graphic printing, only certain types of printers can take advantage of this option. See your computer's Spec Sheet for more information on installing printer drivers, and what type of printer is supported for your computer.

Save Puzzle & Load Puzzle

These options allow you to save and load the words in the word list along with any puzzle in the Puzzle Box. The file name used when saving or loading a puzzle becomes the puzzle's title. For details on legal file names, see your computer's Spec Sheet.

Future Plans

Well, all good programs should leave room for expansion, and this is where we left room in *Crossword Composer*. "What type of expansion?" you ask. Maybe the next volume of HCJ will tell. Anyway, the program's screen just wasn't as attractive with only 7 options.

Exit Program

Be sure to save your puzzle before quitting. For safety's sake, however, you are asked if you are sure that you want to quit before the program aborts.

Sample Puzzles

To get you started, we have provided a puzzle of our own on the HCJ Volume 3 disk. To load this puzzle, select the Load option in *Crossword Composer* and enter GUITAR as its file name. Can you solve this puzzle without looking at the solution?

Crossword Composer Spec Sheet: C-64

Control Capsule
Edit Word Mode

KEY	FUNCTION
RETURN	Move to clue or next word
Cursor down	Move to clue or next word
Cursor up	Move to word or previous clue
F1	Move to previous word/clue
F3	Move to next word/clue
F5	Move back by 5 words
F7	Move forward by 5 words
F2	Cut word/clue from list
F4	Paste word/clue into list
Back arrow	Return to menu

To run the C-64 version of *Crossword Composer*, simply **LOAD** and **RUN** the program file CROSSWORD. You may also use the HCJ Director program.

Printer Drivers

There are two printer drivers on the C-64 HCJ Volume 3 disk:

File name	Printer
`ALL.PRT`	Any printer —no graphics
`EPSON.PRT`	Epson-compatible dot-matrix printers

The program file CROSSWORD has the ALL.PRT driver installed and the program file CROSSWORD.EPSON has the EPSON.PRT driver installed. So, you can just load and run the program of your choice—the only difference being printer drivers.

Although we provide versions of *Crossword Composer* with the different printer drivers already installed, it is possible to install a driver yourself. To install a driver, follow these steps:

1) **LOAD** the CROSSWORD program. For example:

```
LOAD "CROSSWORD",8
```

2) **LOAD** the printer driver of your choice (using a **,1** extension). For example:

```
LOAD "EPSON.PRT",8,1
```

3) Execute a **SYS 3558** and **SAVE** the program under the file name of your choice. For example:

```
SYS 3558
SAVE "CROSSWORD.EPSON",8
```

Because the HCJ disk is write-protected, you must use a backup of the disk to complete this procedure.

File Names

File names for puzzles can be up to 12 characters in length. When saved to disk, your file will appear in the directory with a .PUZ extension. Puzzles are saved as program files, but they are not to be loaded as such!

Crossword Composer Spec Sheet: IBM

Control Capsule
Edit Word Mode

KEY	FUNCTION
ENTER	Move to clue or next word
Cursor down	Move to clue or next word
Cursor up	Move to word or previous clue
F1	Move to previous word/clue
F2	Move to next word/clue
Page up	Move back by 5 words
Page down	Move forward by 5 words
F3	Cut word/clue from list
F4	Paste word/clue into list
ESC	Return to menu

To run the IBM version of *Crossword Composer*, simply enter **CROSS** at the DOS prompt. You may also use the HCJ Director program.

Printer Drivers

There are two printer drivers on the IBM HCJ Volume 3 disk:

File name	Printer
`IBM.DRV`	IBM compatible dot-matirx printers
`EPSON.DRV`	Epson compatible dot-matrix printers

If you do not use one of the above printer drivers, this program prints the puzzle using asterisks for the puzzle boxes, and is compatible with all printers. To use one of these printer drivers, all you have to do is copy the printer driver of your choice to the file name PRINTER.DRV. This can be done from DOS using the following command:

```
COPY IBM.DRV PRINTER.DRV
    or
COPY EPSON.DRV PRINTER.DRV
```

Because the HCJ disk is write-protected, you must use a backup of the disk to complete this procedure.

Basically, when *Crossword Composer* is first run, it searches the disk for the PRINTER.DRV file, and if found, it installs that printer driver. For this reason, the PRINTER.DRV file must be in the active drive when *Crossword Composer* is run, or the driver will not get installed. For ease of use, we suggest that you copy your printer drivers onto the same disk as the crossword program.

File Names

File names for saving and loading puzzles can be up to 8 characters in length. When saved to disk, your file will appear in the directory with a .PUZ extension.

Crossword Composer Spec Sheet: TI-99/4A

Control Capsule
Edit Word Mode

KEY	FUNCTION
ENTER	Move to clue or next word
FCTN X	Move to clue or next word
FCTN E	Move to word or previous clue
CTRL 4	Move to next word/clue
CTRL 6	Move to previous word/clue
FCTN 4	Move forward by 4 words
FCTN 6	Move back by 4 words
CTRL 1	Cut word/clue from list
CTRL 2	Paste word/clue into list
FCTN 9	Return to menu

To run the TI-99/4A version of *Crossword Composer*, simply **OLD** and **RUN** the program file CROSSWORD. You may also use the HCJ Director program. This program uses upppercase letters only, so it is advised that you keep the **[ALPHA LOCK]** key down at all times.

Printer Driver

When you run the program file CROSSWORD, it will default to using asterisks for drawing boxes on any printer. If you prefer a more elegant graphic output, and own an Epson-compatible printer, you can use the EPSON_DRV printer driver.

To use the Epson printer driver, all you have to do is copy the file EPSON_DRV to the new file name PRINTER_DRV. This can be done using the Disk Manager's Copy File option. Because the HCJ disk is write-protected, you must use a backup of the disk to complete this procedure.

Basically, when *Crossword Composer* is first run, it searches the disk for the PRINTER_DRV file, and if found, it installs that printer driver. For this reason, the PRINTER_DRV file must be in DSK1 when *Crossword Composer* is run, or the driver will not be installed. For ease of use, we suggest that you copy the printer driver onto the same disk as the crossword program.

File Names

File names for saving and loading puzzles can be up to 8 characters in length. File names must be preceded by the device (i.e., DSK1.). When saved to disk, your file will appear in the directory with a _X extension.

Perfect Puppy

You've heard of Pavlov's dogs, but have you heard of HCJ's Perfect Puppy? Just like those obedient canines, our own cyber-pup learns from both positive and negative reinforcement.

Can a computer learn? Of course it can. Just type in a program and the computer will do just what you've told it to. But, is that *intelligence?* Just because a computer can be programmed to play a game, doesn't mean it has really learned anything. Discussions of this type can be heard in computer circles around the world, and in educational institutions—from grade school to graduate school.

To help you get some first-hand experience with these issues, we've designed a program to show one way a computer can be programmed to learn—by allowing it to observe correct responses to given situations, and be able to respond in a like manner when confronted with those situations. Because the program starts with a blank slate, and learns *only* exactly what you teach it—no matter how good or bad a teacher you are—we call it the *Perfect Puppy.*

Before we get into the explanation, realize that this program does *not* pretend to demonstrate the latest in artificial intelligence, nor is it the *only* way to teach a computer to play a game. It is, however, *one* way to show how the computer—with some easy-to-understand BASIC—can simulate a learning process. In fact, if you *are* a good teacher, you can teach the *Perfect Puppy* to win nearly every game it plays.

The Game

The game we've chosen to teach our cyber-pup is relatively simple—but, if you've never played it before, requires enough strategy to be quite challenging. The game plays an important part in a 1961 French movie called "Last Year at Marienbad." One of the characters in the movie states that he knows a game he can always win.

"If you can't lose," his opponent counters, "it's not a game!"

"I can lose," he responds. ". . . But I always win."

The game takes two to play. Sixteen match sticks are arranged in rows of one, three, five, and seven. Each player picks up sticks in turn (as many sticks as he wants) on the condition that he takes from only one row each time. The one who picks up the last stick has lost. We designate the rows A, B, C, and D. Thus, you move by choosing one of the four letters to choose a row, and then selecting the number of sticks you wish to remove. When the program runs for the first time, it knows the rules of the game, and properly declares the winner at the end. But, because it has no knowledge of a good or bad move (providing that you don't first load in any experience), it chooses its moves at random.

Running The Program

When you run *Perfect Puppy,* you are first asked if you wish to load the computer's memory with game experience. As the computer plays, it notes the moves that lead to a win or a loss, and this knowledge guides its own future play. This knowledge can be saved to disk at the end of any session, and can be reloaded the next time the program is run. By giving these files different names, you can have several knowledge bases to experiment with. We've included one file called SMARTS.AI for you to get started with. Note that the .AI extension is added by the program, and should *not* be entered by the user.

Next you are presented with the main menu:

```
1) HUMAN MOVES FIRST
2) COMPUTER MOVES FIRST
3) HUMAN VS HUMAN
4) COMPUTER VS COMPUTER
5) AUTO-LEARN MODE
6) QUIT
```

In the first two options, you play against the computer. If you have not loaded a knowledge file, the computer plays totally at random, choosing any available move. Against a reasonably aware opponent, the computer nearly always loses. It *can* win, but it rarely does.

Beating the computer, however, is *not* the idea. Rather, it is your job to improve the computer's game by teaching it the best moves, and *not* teaching it bad habits. Like a puppy, the program mimics moves that are rewarded by winning, and tends to shun moves that lose. If you play a game perfectly, only to make a foolish move at the end, your puppy views all of your moves in that game as bad moves, and your teaching task will be made harder. This is analogous to punishing your puppy when he scratches your freshly painted door while signaling his desire to go outside to do his business.

Option 3, Human vs. Human, allows you to play against another person, or teach your puppy certain patterns of moves by controlling both sides. The computer never stops observing and modifying its knowledge base, so even though it is not actively in the game, its expertise will improve if it "watches" two good players.

Option 4, Computer vs. Computer, pits the computer against itself. This is a good observing mode, so you can check to see if there is some response in your puppy's knowledge base that you have not noticed when playing against it yourself. The computer might even teach you a thing or two in this mode.

In option 5, Auto-Learn Mode, the computer and an opponent (Mr. Random) take turns going first, and keep playing until they are stopped. To stop Auto-Learn Mode, press [ESC] (back-arrow on the C-64, [FCTN] 9 on the TI). This mode is excellent for getting a knowledge base started, because the computer is eventually faced with nearly every situation. But keep in mind that Mr. Random does *not* necessarily make the best move; therefore, even after several hours of this mode, your puppy will need some careful tutoring to become a really good player.

When you Quit, option 6, be sure to save the computer's knowledge so you won't have to start from scratch when you start the next session.

Game Notation

The current number of match sticks in each row (the board position) is kept in the string variable **BD$**. Every time a move is made, **BD$** is updated to reflect the new board position. The board position is stored as 4 characters. The first character reflects the number of match sticks in row A, the second character reflects the number of match sticks in row B, and so on.

The initial board position (1 stick in row A, 3 sticks in row B, 5 sticks in row C, and 7 sticks in row D) is represented as "1357" in **BD$**. If you were to remove 3 sticks from row B in your first move, "1057" would be the resulting board position. The four board positions "1000", "0100", "0010", and "0001" signify the end of the game.

In order for our puppy to learn from a game, it must keep track of the game's moves. Each move in a game is stored in the string array **GM$( )**. The first move of the game is stored in **GM$(1)**, the second move in **GM$(2)**, and so on. The current move number is kept in the variable **MV**.

Moves are stored in a seven-character format. The first four characters represent the board position (see above). The fifth character is always a space. The sixth character represents the row (A-D), and the seventh and last character specifies the number of match sticks removed. If you were to remove 4 match sticks from row C in the first move of the game, for example, the string notation "1357 C4" would be placed into **GM$(MV)**.

The total number of moves in a game is recorded in the variable **NM**. If you wish to list all of a game's moves, you could do so by breaking the program after a game and entering the following code:

```
FOR I=1 TO NM :: PRINT GM$(I) :: NEXT I
```

The double colons are only necessary for TI-99/4A computer owners. Both double and single colons are acceptable command separators on other machines.

Dog Brains

Perfect Puppy learns from experience: Good behavior (winning) is rewarded, and bad behavior (losing) is punished. (Note: Here, you can forget about that noble maxim "It's not whether you win or lose . . .") In order for *Perfect Puppy* to remember what is good and bad behavior, we had to supply our young canine with a memory or brain. This brain is provided by the string array **WM$( )**. Winning moves are moved into this array, while losing moves are removed. At the start of the program, this array is empty.

Each array element in **WM$( )** makes up a brain cell, and each brain cell keeps track of the winning moves for a particular board position. Indexing into a brain cell is done in a very simple matter. For example, winning moves for board position "1354" are stored in **WM$(1354)**, while winning moves for board position "0211" are stored in **WM$(211)**. So, at any time during the game, **WM$(VAL(BD$))** returns the winning moves (if any) for the current board position. (One point of interest: With 383 possible board positions and 1357 brain cells, only about 3% of *Perfect Puppy's* brain cells are actually ever used. Remind you of any dogs you've met?)

Winning moves are stored in **WM$( )** in the same format as game moves are stored in the last two characters of the **GM$( )** array with the first character representing the row, and the second character representing the number of match sticks to be removed. Several moves may be stored in **WM$( )**. For example, the board position indicated by **WM$(1536)** may contain "C3B1A1C1B2" as its winning moves. This example offers the 5 winning moves C3, B1, A1, C1, and B2. It is very possible that there will be just one, or no winning moves stored for any board position.

Brain cells organize moves in order of preference: Superior moves appear ahead of inferior moves. Using the previous example, we see that our *Perfect Puppy* prefers the move C3 over B1, B1 over A1, A1 over C1, and C1 over B2. Hoping to make the correct response, *Perfect Puppy* always uses the *first* move found in a brain cell.

If you wish to delve further into *Perfect Puppy's* inner workings, see Key Variables and the accompanying sidebar, Learning To Win.

Key Variables

Variable	Function
WM$( )	Stores winning moves
GM$( )	Stores current game's moves
BD$	Current board position
PL	Current player (0 or 1)
MV	Current move number
NM	Total number of moves

Learning To Win

Perfect Puppy evaluates each game, trying to learn some winning moves. This learning process involves looking at *both* player's moves, and updating the **WM$( )** array (*Perfect Puppy's* brain) accordingly.

Each move made by the winning player is placed into the brain cell corresponding to the board position in which the move was made. Consider the case when the winning player removes 1 stick from row A in the following board position:

```
A) |
B) ||
C) ||
D)
```

The string "A1" is placed into **WM$(1220)**. If **WM$(1220)** already contains the move A1, then the A1 move is bumped up by one move in the list. If the move A1 is already first in the list, then no change is made. If A1 is not already in the list, then the move A1 is placed as the first move in the list. Here are some before-and-after examples:

	Before	After
WM$(1220) =	""	"A1"
WM$(1220) =	"B1B2A1C1"	"B1A1B2C1"
WM$(1220) =	"A1B1B2"	"A1B1B2"
WM$(1220) =	"B1B2"	"A1B1B2"

Each move made by the losing player is removed from the brain cell corresponding to the board position in which the move was made. Consider the case when the losing player removes 2 sticks from row C in the following board position:

```
A) |
B) |||
C) |||
D) |||||
```

The string "C2" is removed from, or lowered in importance from **WM$(1335)**. If **WM$(1335)** already contains the move C2, then C2 move is bumped down by one move in the list. If the move C2 is already last in the list, then C2 is removed from the list. If C2 is not already in the list, then no change is made. Here are some before-and-after examples:

	Before	After
WM$(1335) =	"D4C2A1"	"D4A1C2"
WM$(1335) =	"D4A1C2"	"D4A1"
WM$(1335) =	"A1D4B2"	"A1D4B2"

ACCEPT AT (New & Improved)

If you were to make a list of Extended BASIC's top ten commands, chances are that **ACCEPT** would be included. This command provides a safe-yet-convenient means of receiving input. The TI Extended BASIC manual explains this command's capabilities best: "Many options are available with **ACCEPT**, making it far more versatile than **INPUT**. It may accept data at any screen position, make an audible tone (beep) when ready to accept the data, erase all characters on the screen before accepting data, limit data accepted to a certain number of characters, and limit the type of characters accepted."

So, what's better than the **ACCEPT** command? Our new-and-improved **ACCEPT** command, of course. This new version is provided on your HCJ Volume 3 disk under the file name ACCEPT_O. It is an assembly-language object file that is loaded by the following commands: `CALL INIT :: CALL LOAD("DSK1.ACCEPT_O")`

For this code to function properly, you must have a disk in drive 1 that contains the ACCEPT_O file. You must also have TI Extended BASIC and the 32K Memory Expansion.

Before we explain why this new **ACCEPT** command is better, let's discuss the limitations found in the standard **ACCEPT** command. First off, you can only input *one* screen line of text when using **ACCEPT AT**. This is probably the **ACCEPT** command's biggest limitation. (Note: It is possible to use the **ACCEPT** command without the **AT** option and receive up to 255 characters, but the input prompt will appear at the bottom of the screen, causing the screen to scroll when more than 27 characters are entered.)

Next, there are only three keypresses that will exit an **ACCEPT** command—[ENTER], [FCTN] E, and [FCTN] X. What if you want to provide the user with a [FCTN] 9 escape option, or allow them to move from one input field to another through the use of [CTRL] key combinations? With Extended BASIC's **ACCEPT**, you can't. You are stuck with the three keypresses listed above, and no direct way to tell which of the three keypresses was used to terminate input.

This is where our new **ACCEPT** command comes in. Our version of the **ACCEPT** command accepts up to 255 characters *anywhere* on the screen. You can also define up to 15 exit keys and the key that is used to exit the **ACCEPT** command is returned in the string variable of your choice. The format of our **ACCEPT** command is shown in Figure 1.

There are some things that you must take into account when using our new **ACCEPT** command. First, to create a cursor, this program makes special use of the sprite definition table. Consequently, you cannot have any sprites on the screen during execution of the new **ACCEPT** command. If there are any sprites active when you call the new **ACCEPT** command, they will be turned off, just as if you had issued the command **CALL DELSPRITE(ALL)**. You can always re-activate your sprites after using the new **ACCEPT** command by calling them up again with the **CALL SPRITE( )** command.

Because the input cursor is a sprite, it can have a different color than any of the characters. In fact, the cursor should always be a different color than any of the characters, or you will not be able to see the character that the cursor is currently on top of. To specify the color of the cursor, use the following command: `CALL COLOR (#1,COLOR)`

The **COLOR** parameter must be a number between 1 and 16, corresponding to the color of your choice (see Figure 2). This command should precede any calls to the new **ACCEPT** command. If you do not set the cursor's color prior to using the new **ACCEPT** command, the cursor will probably default to color #1 (transparent). When selecting a color, try to choose one that contrasts well with color of the characters and the screen.

Being able to change the cursor's color can come in handy. With this ability, you can color-code your input fields. For example, you can have the cursor turn blue when prompting for numeric data, and red when prompting for letters.

Refer to this Volume's feature program *Crossword Composer* for an example of how to use this new command from within a program. This program uses the new **ACCEPT** command to input the crossword puzzle's words and clues.

Figure 1

```
CALL LINK("ACCEPT",ROW,COLUMN,SIZE,VALIDATE$,EXIT$,ESC$,S$)
```

Parameters:

ROW,COLUMN (numeric) designates the starting position on the screen for input. The **ROW** can be between 1 and 24 while the **COLUMN** can be between 1 and 28.

SIZE (numeric) defines the maximum number of characters that can be entered. This parameter can vary between 1 and 255.

VALIDATE$ (string) defines the characters that can be entered during input. If this string is null, all characters are considered legal.

EXIT$ (string) contains the various keypresses that can be used to terminate input (up to 15 characters). If this string is null, only the [ENTER] key will terminate input.

ESC$ (string) returns the ASCII of the key used to exit the **ACCEPT** command (see previous parameter).

S$ (string) returns the input data.

Notes: This **ACCEPT** command does not clear the input area of the screen, and any characters that are located in the input area become the default input. Also, there are two options found in the standard **ACCEPT** command that are not implemented in our new version: Our new version cannot **BEEP** or **ERASE ALL**. If you must make noise or clear the screen prior to input, then you can always use the **CALL SOUND** and/or **CALL CLEAR** commands first.

Figure 2

Code	Color
1	Transparent
2	Black
3	Medium Green
4	Light Green
5	Dark Blue
6	Light Blue
7	Dark Red
8	Cyan
9	Medium Red
10	Light Red
11	Dark Yellow
12	Light Yellow
13	Dark Green
14	Magenta
15	Gray
16	White

GETSTR & PUTSTR

Create text windows, fast character graphics, instantaneous screen dumps, and more!

One of the TI-99/4A computer's strongest features is graphics. Both TI BASIC and TI Extended BASIC have several built in commands for manipulating screen graphics. Here, we present even two more; **GETSTR** and **PUTSTR**. These multi-functional commands can add life to an otherwise slow program.

GETSTR is what you might call **GCHAR**'s big brother. The difference between the two commands is that **GETSTR** can get up to 255 characters from the screen, while **GCHAR** can only get one. Now, reading multiple characters from the screen no longer requires the use of slow and cumbersome **FOR-NEXT** loops.

PUTSTR is related to the two commands **VCHAR** and **HCHAR**. The difference here is that **PUTSTR** can place up to 255 characters onto the screen, while **VCHAR** and **HCHAR** can only place one (or a repeated number of one) character. Do not confuse this command's capabilities with the **DISPLAY AT** command. With our's, you not only can specify the starting position of output characters, but you can also specify the left, right, top, and bottom borders in which the characters are displayed. Both of these new commands act on any retangular section of the screen. So, you are not limited to just one character or even one screen line.

How To Use Them

Use of these two new commands requires TI Extended BASIC and the 32K Memory Expansion. These commands are written in assembly language and linked to Extended BASIC through the **CALL LINK** command. The file GETPUT_O on your HCJ Volume 3 disk contains the machine language for these two commands. The following code loads this file into memory:

```
CALL INIT :: CALL LOAD("DSK1.GETPUT_O")
```

You should include this code at the beginning of any program that uses the **GETSTR** or **PUTSTR** command. The **CALL INIT** command only has to be executed once, so if you are going to load any addition assembly language files, such as ACCEPT_O (refer to this Volume's TI Technote), use the following commands:

```
CALL  INIT  ::  CALL  LOAD("DSK1.GETPUT_O")::
CALL LOAD("DSK1.ACCEPT_O")
```

Figure 1

GETSTR Syntax:
```
  CALL LINK("GETSTR",TOPROW,TOPCOL,BOTROW,BOTCOL,S$)
```
PUTSTR Syntax:
```
  CALL LINK("PUTSTR",TOPROW,TOPCOL,BOTROW,BOTCOL,S$)
```

The Syntax

Once GETPUT_O is loaded, you can start using the new commands. Figure 1 shows the syntax for each command.

Both of these commands require that you specify the area of the screen that you wish to get or put characters to. We call this "area of the screen" the window. A window's size and location is defined by its upper-left and lower-right corners. The parameters TOPROW and TOPCOL specify the uppper-left corner of the window. The parameters BOTROW and BOTCOL specify the bottom-right corner of the window. TOPROW and BOTROW must be between 1 and 24 while TOPCOL and BOTCOL must be between 1 and 32.

In a **GETSTR**, the S$ parameter receives characters from the window. If the window area contains more than 255 characters, only the first 255 characters are placed into S$.

In a **PUTSTR**, the contents of S$ are placed into the window. If the window is not large enough to hold the entire contents of S$, only the characters that fit are placed onto the screen. If the window area is larger than S$, the remaining section of the window is filled with blanks.

A Couple Of Examples

We have written two short demonstration programs that take advantage of the **GETSTR** and **PUTSTR** commands. These programs are *DEMO1* and *DEMO2*. Both programs are included on your HCJ Volume 3 disk.

DEMO1 shows how the **GETSTR** and **PUTSTR** commands can create movable, expandable, and updatable text windows on your screen. This program creates a text window that displays the program's instructions. By using the **E**, **S**, **D**, and **X** keys, you can move the windows upper-left corner. The **I**, **J**, **K**, and **M** keys move the lower-right corner of the window. The speed at which the text is re-flowed inside the window is quite remarkable. Let's see you do this one with **HCHAR**s. . .

Besides moving a text window around the screen, you can print the screen by pressing **[CTRL] P**. Normally, one must **GCHAR** the screen, slowly sending each character to the printer. With **GETSTR**, however, *DEMO1* is able to grab a whole screen line at a time, printing the screen in record time. This screen dump subroutine is located in lines 520-590. Press **[FCTN] 9** to exit the program.

Another great use for the **GETSTR** and **PUTSTR** commands is character graphics. There is no quicker method of placing character graphics on the screen than **PUTSTR**. The program *DEMO2* drives this point home. This program displays a screen of text describing the program's operation while a smiling face hyperactively jumps around the screen. You can speed up the frantic face by holding down the **[FCTN] E** key. Holding down **[FCTN] X** calms the smiling face down a bit. To create the face, the **PUTSTR** command places 9 redefined characters on the screen in a three by three block (window).

Notice that characters covered by the smiling face are restored when the face moves (just like sprites!). **GETSTR** saves the area of the screen that the smiling face is about to invade and **PUTSTR** puts the characters back when the face leaves. This speed in character animation is invaluable when designing games and other graphic intensive programs.

Use of these two new commands are not limited to the examples provided here. How about a text editor with visual cut and paste, a character-graphics jigsaw puzzle, or even pull-down menus? No matter what the application may be, we think that you'll find these two commands very useful.

CONTROL CAPSULE

DEMO1

Key	Function
E	Move top corner of window up
X	Move top corner of window down
S	Move top corner of window left
D	Move top corner of window right
I	Move bottom corner of window up
M	Move bottom corner of window down
J	Move bottom corner of window left
K	Move bottom corner of window right
FCTN 9	Exit program

CONTROL CAPSULE

DEMO2

Key	Function
FCTN E	Speed up smiling face
FCTN X	Slow down smiling face
FCTN 9	Exit program

LINE ANNOTATIONS

DEMO1

Line Nos.	Explanation
100-180	Program header
190-200	Load machine-language file
210-300	Initialize program
310-430	Program's main loop
320	Print text window
330	Read keyboard
340	Move top corner left
350	Move top corner right
360	Move top corner up
370	Move top corner down
380	Move bottom corner up
390	Move bottom corner down
400	Move bottom corner left
410	Move bottom corner right
420	Initiate screen dump
430	Check for [FCTN] 9
440-510	Exit program
520-590	Print screen dump

LINE ANNOTATIONS

DEMO2

Line Nos.	Explanation
100-180	Program header
190-200	Load machine-language file
210-250	Initialize program
260-330	Print instructions
340-410	Program's main loop
350	Get random screen position
360	Get underlying characters and print face
370	Read keyboard and check for [FCTN] 9
380	Speed up or slow down faces's speed
390	Delay loop
400	Restore underlying characters
410	Go to beginning of loop
420-480	Exit program
490-510	Character graphics data

IBM PC, PCjr, and Tandy 1000

Procedures For Using The IBM PC, PCjr, Or Tandy 1000

To make use of the HCJ Director menu program on your HCJ disk you need to backup your disk. Use the following procedures to produce an autoboot backup of your HCJ disk:

If you have a dual-drive system you may start with step 1, otherwise read this paragraph first:

For those of you with a single disk drive, you may still use the commands as listed below, though you will need to pay very close attention to the prompts on the screen instructing you to swap disks from time to time. The computer will tell you to place the appropriate disk in drive B:. What it means, however, is to remove the disk from drive A: and insert the disk which would have gone in drive B:. Using a single drive may mean having to swap disks quite a few times. For those who are patient though, the rewards are worth the added work. If you have further questions consult your DOS manual on the FORMAT and COPY procedures for a single-drive system.

1. Place your DOS master disk (hereafter refered to as the DOS disk) in drive A: and turn on the power to your system.
2. Enter the command **FORMAT B: /S /V**
3. The computer will ask you to place a blank disk into drive B: to be formatted. Ensure that the blank disk is in the drive and then press [Enter]. After formatting, you will be asked for a volume name. Enter **HCJOURNALn** where **n** is the Volume number. Then, you will be asked if you want to format another. Respond No to this prompt which returns you back to DOS.
4. If you wish to use a color monitor enter the command **COPY A:MODE.COM B:**
5. Enter the command **COPY A:BASIC*.* B:BASIC.***
If you have an IBM compatible whose BASIC does not start with the word BASIC, then make adjustments in the command above for your version. In any case, the file on your new boot disk should always be named BASIC even if it was originally named BASICA.
6. After BASIC is copied to the new disk in drive B:, remove the DOS disk from drive A: and place the HCJ disk in drive A:
8. Enter the command **COPY A:\*.* B:**
9. After the last file has been copied, remove both disks from the system. Label the new disk as **HCJ ON DISK BACKUP Volume n**, where **n** is the Volume number, and place the original disk in a safe place.
10. The new disk you have created can now be used to boot your system (start from a power off condition) and will automatically bring up a menu of programs from which you may select.

You may also use the HCJ disk without backing it up if you:
1. Start from DOS 2.1 or later.
2. If you wish to run a program with a BAT, COM, or EXE extension, simply type the file name from the DOS A> prompt.
3. If you wish to run a BASIC program, you must first enter the appropriate version of BASIC, then **LOAD** and **RUN** the program.
Note: If you have an IBM PC and the program requires a color monitor, you must enable the monitor using the appropriate DOS **MODE** command before running the program.

Program Name	**File Name**	**Language**
HCJ Director	HCJDIR.COM*	Turbo Pascal
	AUTOEXEC.BAT	-batch file-
CodeWorks	CODEWORK.COM*	Turbo Pascal
IS-Base	IS-BASE.COM*	Turbo Pascal
	CAPITALS	-data file-
	MADONNA	-data file-
Crossword Composer	CROSS.COM*	Turbo Pascal
	IBM.DRV	-printer driver-
	EPSON.DRV	-printer driver-
	GUITAR.PUZ	-data file-
Perfect Puppy	PERFECT.BAS**	BASICA
	SMARTS.AI	-data file-
Expanded DOS	CFD.COM*	Turbo Pascal
	HIDE.COM*	Turbo Pascal
	SHOW.COM*	Turbo Pascal
	REVEAL.COM*	Turbo Pascal
FormFlex	FORMFLEX.COM***	Turbo Pascal
	MAKEFORM.CHN***	Turbo Pascal
	USEFORM.CHN***	Turbo Pascal

*Program requires: DOS 2.1 or later.
**Program requires: DOS 2.1 & either Cartridge BASIC on PCjr or BASICA on PC, or GW BASIC on Tandy 1000.
***The original version of *FormFlex* provided on the HCJ Volume 2 disk does not run on some PCjr systems (insufficient memory often being the result). The version of *FormFlex* provided here has been has been modified to run on any 128K PCjr.

TI-99/4A

Procedures For Loading The TI-99/4A With Extended BASIC

1. Ensure the Peripheral Box is properly connected to the console. Turn on the Peripheral Box.
2. Place the Extended Basic module securely in the machine.
3. Turn on the TI-99/4A computer.
4. Insert the HCJ disk into drive 1.
5. Strike any key to bring up the first menu, then select Extended BASIC, and The HCJ Director program will automatically **RUN**.
6. Select the number of the program you wish to run, then press **[ENTER]** and the program will load and **RUN** automatically.

Procedures For Loading The TI-99/4A With TI BASIC

1. Ensure the Peripheral Box is properly connected to the console. Turn on the Peripheral Box.
2. Turn on your computer and insert the HCJ disk in drive 1.
3. Strike any key to bring up the first menu, then select BASIC.
4. To load the BASIC program you wish to use, type **OLD DSK1.file name** where **file name** is the file name of the program. For example, if you wish to use *Perfect Puppy* type **OLD DSK1.PERFECT** and press [ENTER]. Now, type **RUN** and press **[ENTER]**.

***

Due to the extensive size and number of this Volume's programs, we were unable to include the *Codeworks* programs on your HCJ Volume 3 disk—it just wouldn't fit! We plan to include *Codeworks* on all future HCJ disks.

***

Program Name	**File Name**	**Language**
HCJ Director	LOAD	Extended BASIC
IS-Base	IS-BASE	Extended BASIC
	CAPITALS	-data file-
	MADONNA	-data file-
Crossword Composer	CROSSWORD*	Extended BASIC
	CROSS_O	TMS 9900 object file
	ACCEPT_O	TMS 9900 object file
	EPSON_DRV	-printer driver-
	GUITAR_X	-data file-
Perfect Puppy	PERFECT	BASIC
	PERFECTX	Extended BASIC
	SMARTS_AI	-data file-
GETSTR & PUTSTR	GETPUT_O*	TMS 9900 object file
	DEMO1*	Extended BASIC
	DEMO2*	Extended BASIC
ACCEPT (improved)	ACCEPT_O*	TMS 9900 object file

*Requires 32K Memory expansion.

HCJournal™

Don't Miss A Single Volume
— Subscribe/Renew Today...
...And Don't Forget
To Tell Your Friends!

Home Computing Journal (HCJ) is a quarterly multi-media software subscription service containing ready-to-run productivity, education, entertainment, and utility programs on a floppy disk. The accompanying workbook contains the required support documentation plus additional technical notes and programming aids.

Artificial intelligence, database management, high-powered programming aids, realistic simulations, and specialized software for personal investing, task-specific report writing, computer-assisted design, desktop publishing, personal communications, plus entertaining math and logic excursions are just some of the projects already on our planning board.

YES, Please accept my order for

Home Computing Journal:

☐ New subscription ☐ Renewal subscription

☐ 2-Volume Mini-Subscription: **$45** postpaid U.S. / U.S.**$52** in Canada

☐ 4-Volume Subscription **$75** postpaid U.S. / U.S.**$89** in Canada

☐ Single-Volume Price: **$25** postpaid U.S. / U.S.**$30** in Canada

Each quarterly Volume of HCJ includes the printed Journal plus the companion disk for your computer choice indicated below.

DISK VERSION					
APPLE	ATARI	C-64	IBM PC	IBM PCjr	TI

Please start with Volume No. ☐

Each Volume is numbered sequencially—i.e., Volume 1, Volume 2, Volume 3...

TOTAL ORDER: ____________

☐ Check or Money Order Enclosed
MUST BE IN U.S. FUNDS DRAWN ON A U.S. BANK

Charge my: ☐ VISA ☐ MasterCard

Account No. ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

DATE EXPIRES __________ SIGNATURE ____________________

PLEASE PRINT ALL INFORMATION BELOW

NAME ____________________

ADDRESS ____________________

CITY ____________________ STATE ____ ZIP ________

TEL. NO. ____________________

Prices Subject To Change Without Notice.

Mail with check, money order, or credit card information to:

Home Computing Journal
P.O. Box 70248 • Eugene, OR 97401

MC/VISA orders may also be placed by telephone:

Tel. (503) 342-4013

West Coast Time (Normal Business Hours)

If you prefer not to cut this copy of your Journal, you may photocopy this form to send in with your order.

HCJournal™

Don't Miss A Single Volume
— Subscribe/Renew Today...
...And Don't Forget
To Tell Your Friends!

Home Computing Journal (HCJ) is a quarterly multi-media software subscription service containing ready-to-run productivity, education, entertainment, and utility programs on a floppy disk. The accompanying workbook contains the required support documentation plus additional technical notes and programming aids.

Artificial intelligence, database management, high-powered programming aids, realistic simulations, and specialized software for personal investing, task-specific report writing, computer-assisted design, desktop publishing, personal communications, plus entertaining math and logic excursions are just some of the projects already on our planning board.

YES, Please accept my order for

Home Computing Journal:

☐ New subscription ☐ Renewal subscription

☐ 2-Volume Mini-Subscription: **$45** postpaid U.S. / U.S.**$52** in Canada

☐ 4-Volume Subscription **$75** postpaid U.S. / U.S.**$89** in Canada

☐ Single-Volume Price: **$25** postpaid U.S. / U.S.**$30** in Canada

Each quarterly Volume of HCJ includes the printed Journal plus the companion disk for your computer choice indicated below.

DISK VERSION					
APPLE	ATARI	C-64	IBM PC	IBM PCjr	TI

Please start with Volume No. ☐

Each Volume is numbered sequencially—i.e., Volume 1, Volume 2, Volume 3...

TOTAL ORDER: ____________

☐ Check or Money Order Enclosed
MUST BE IN U.S. FUNDS DRAWN ON A U.S. BANK

Charge my: ☐ VISA ☐ MasterCard

Account No. ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

DATE EXPIRES __________ SIGNATURE ____________________

PLEASE PRINT ALL INFORMATION BELOW

NAME ____________________

ADDRESS ____________________

CITY ____________________ STATE ____ ZIP ________

TEL. NO. ____________________

Prices Subject To Change Without Notice.

Mail with check, money order, or credit card information to:

Home Computing Journal
P.O. Box 70248 • Eugene, OR 97401

MC/VISA orders may also be placed by telephone:

Tel. (503) 342-4013

West Coast Time (Normal Business Hours)

If you prefer not to cut this copy of your Journal, you may photocopy this form to send in with your order.