Holmes & Duckworth

micronomists

H & D Base

Atari 520ST®

Holmes & Duckworth

H & D Base

USER'S MANUAL

micronomists

MIRAGE
4055 W. Shaw. #108
Fresno, CA 93711
(209) 227-8369

Created by

Chester Holmes & Oliver Duckworth

REV. A, January 1986

A Note From the Authors

I am quite pleased (and honored) to have the opportunity of speaking with you once again. Your gestures of trust, as demonstrated through the continuing support of our programmes, have proven to be most undeniably encouraging to both me and my co-conspirator, Mr. Chester Holmes.

It was encouragement of this nature which inspired Holmes and me months ago to take on our most ambitious project to date. Our feeling was that a substantial relational database language *had to be developed* for the Atari ST line of computers if they were ever to establish their rightful position of usefulness in the business environment.

In talking with many of you about a package of this type, we found that the name of one long-established product came up time and time again. The product? "dBASE II" -- a name which has become almost synonymous with the term "relational database language" in the vocabulary of computer users around the world. Even now, after an initial release that occurred over four years ago, thousands upon thousands of "dBASE" programs are sold *every month*.

Why has "dBASE II" been so phenomenally successful? There are, undoubtedly, a number of reasons, not the least of which are that it was the first program of its type, that it is both powerful and relatively easy to use, and that it is easily adaptable to an unlimited number of applications (thousands are commercially available).

It quickly became apparent that any program we were to develop should duplicate many of the fine qualities of "dBASE II", and include, if possible, the facilities for executing "dBASE II" command and data files. In addition, we were convinced that the language could be appreciably improved, especially in light of the unique capabilities afforded with the new Atari machines.

With these criteria in mind, we began months ago to develop the code that would eventually become **H & D Base**. I must say at this point that it was one of the most difficult, and therefore rewarding tasks Chester and I have ever undertaken.

We chose to write **H & D Base** in the FORTH programming language. We did so because the structure of FORTH most closely approximates that of the relational database language we envisioned. We used as a basis the 200 commands which are part of **H & D Forth** (written by Chester and I and released by Mirage in 1985).

Authors' Note, cont.

To the 200 original FORTH commands, we eventually added approximately 75 more during the programming of **H & D Base**. These 75 actually form the core of **H & D Base**, and are the ones which are detailed in this manual. The remainder are documented in a file on the Program Disk and can be used by professional programmers (who have purchased the "Developer's Kit" from Atari).

If you are already familiar with "dBASE II," you will find **H & D Base** to be very similar in regards to programming with it. It is comprised of the best features of versions 2.3 and 2.4, and corrections have been made to some of the most critical shortcomings characteristic of *all* versions of the "dBASE II" program.

Chester and I have developed **H & D Base** to a point where it most assuredly matches, and, in many cases, surpasses the capabilities of "dBASE II." I find it necessary to say, however, that we do not for a moment presume that our task has been completed. We have every intent of continually tinkering with the program in an effort to enhance its performance.

Editor's Note: We will make every effort to keep you informed of any enhancements made by Messrs. Holmes & Duckworth to **H & D Base**. *Please return your "Program Registration Card" so that we will have your address on file for this purpose.*

Holmes, by nature, is more content with lavishing in the laboratory in an attempt to establish absolute programming perfection than he is with taking a few short moments of time to place his thoughts on paper. The duty of documenting the procedures for properly executing this package has fallen, therefore, upon my shoulders. I do hope that I have been equal to the task.

Please drop a note with your comments on the programme to either Chester or myself in care of Mirage. We will make every effort to reciprocate.

As always, I wish you the very best...

Oliver Duckworth

CONTENTS

Table of Contents, cont.

Table of Contents, cont.

Table of Contents, cont.

Table of Contents, cont.

Table of Contents, cont.

Table of Contents, cont.

Chapter 1 INTRODUCTION

The Program

A Relational Database Language

H & D Base is a relational database management language for the Atari ST series of computers. As an extemely powerful information handling tool, the program easily adapts to the needs of both casual and concentrated computer users.

Because **H & D Base** is a *relational* database, it is capable of more than simply filing and retrieving. A relational database program can compare, contrast, and manipulate the fields and records of multiple files which are relative to one another.

The Nature of H & D Base

Keep in mind that **H & D Base** is a database management **language**. When you arrive at its "Command Level," there are no screens or menus -- only a prompt which reads "ok". From this point, it is your task to build the files, screens, menus, and programs that fulfill your particular need. The process can be as simple or as complex as you desire it to be; it's all up to you. Programming with **H & D Base** can be fun, rewarding, and even profitable!

System Requirements

- Atari 520ST Computer
- Atari SF354 Single Sided Disk Drive (1 or 2) or
  Atari SF314 Double Sided Disk Drive (1 or 2) or
  Atari SF317 Hard Disk Drive
- Atari SC124 Monochrome Monitor or
  Atari SC1224 Color Monitor (Low or Medium Res.)
- Printer (Optional)

The Program, cont.

Specifications

Records per File	Limited Only By Disk
Characters per Record	2000 Maximum
Fields per Record	97 Maximum
Characters per Field	254 Maximum
Numeric Accuracy	8 Digits
Character String Length	254 Characters Max.
Command Line Length	254 Characters Max.
Report Header Length	254 Characters Max.
Index Key Length	100 Characters Max.
Expressions in SUM Command	5 Maximum

The Program Serial Number

A program serial number has been stamped on the label adhered to your Program Disk. The number allows us to instantly identify what program you are using, when it was released, and what version it is. DO NOT REMOVE THIS LABEL OR ALTER THE SERIAL NUMBER. IT WILL VOID YOUR WARRANTY.

When communicating with Mirage Concepts concerning your program, always refer to the serial number on the Program Disk.

Copying the Program: Your Rights and Ours

The **H & D Base** Program Disk is *not* copy protected in any manner. We expect and encourage you to make as many copies as you wish, provided they are *for your own personal use only*.

Please honor the rights of this company and the programmers we represent by making every effort to see that unauthorized copies of the program do not fall into the hands of those who have not paid for the right of using them.

If you are aware of any person distributing unauthorized copies of **H & D Base**, please notify us immediately at (800) 641-1441 or (800) 641-1442 in California. For calls originating outside the US: (209) 227-8369.

For policies and prices on multiple copies of **H & D Base** for corporate or educational use, contact Mirage Concepts.

Replacing the Program

If the program disk fails to perform properly at any time, and the problem can be isolated to the disk itself, a new Program Disk will be issued to you UPON RECEIPT OF YOUR DEFECTIVE ONE. There is no charge for this service if the program is in warranty, but a copy of your sales receipt must accompany the defective disk in order to verify the date of purchase. A service charge of $10.00 must accompany any Program Disk out of warranty.

Registering the Program

If you would like information sent to you regarding any enhancements which are made to **H & D Base** fill-out the "Software Registration" card included in the package and send it directly to Mirage Concepts.

Supporting the Program

For answers to special questions regarding this, or any Mirage Concepts software program, customer support personnel are available from 1:00pm to 5:00pm (Pacific) Monday through Friday at (209) 276-1485.

Transferring the Program to a Hard Disk

Because **H & D Base** is not copy protected, you can transfer it to a hard disk drive without fear of malfunction. To do so, simply copy the files ''HDBASE.PRG" and "HDBASE.HLP" from the Program Disk to your hard disk drive using the standard DESKTOP functions. For further information, see "SET DEFAULT" in the "Reference" section.

The Program, cont.

Using Non-H&D Base Files

You may use "dBASE II" files (both COMMAND and DATA) and data files from many other database management programs with **H & D Base**. The instructions for doing so are covered in "Appendix B."

Supplementary Programs

A **H & D Base** COMMAND FILE (program) has been included on the Program Disk. The program is fully operable and may useful to you as you learn how to properly interact with the system.

The COMMAND FILE can be easily spotted from the Atari DESKTOP due to a ".CMD" suffix which follows its name.

The instructions for running a COMMAND FILE are covered at the beginning of the "Programming Tutorial."

The Manual

Organization

This manual provides step-by-step instructions on how to make use of the full capabilities of the **H & D Base** program. It is divided into nine basic sections.

The first of the nine sections, the "Introduction," deals primarily with logistics. It is here that you will find information on what type of computer system you must have to run **H & D Base**, how much data it will handle, what is included in the package, how to receive information on future program updates, and what to do if you have any problems with the disk or the actual **H & D Base** program.

The second section deals with "Starting Up" -- the process of actually getting the program up and running on your particular system. Because of the ease with which the Atari 520ST operates, this section is quite brief.

The first three sections of the manual are the "Beginning, Intermediate, and Advanced Command Tutorials." They are logical, step-by-step training guides for the commands in the **H & D Base** programming language which do *not* deal with creating an actual program or "Command File."

Once you have mastered the correct application of each database command, you will want to proceed to the "Programming Tutorial." It will teach you how to mold **H & D Base** instructions into useful programs (or "Command Files").

The best way to learn how to use **H & D Base** is to review all tutorial sections from beginning to end, taking special care to type-in each example as it is covered.

The "Reference" section, which follows the tutorials,is, perhaps, the most important section of the entire manual. It is the one place you can look for all of the various forms a particular command, function, or operator can take.

The Manual, cont.

The "Tutorial" and "Reference" sections are followed by an "Appendices" which includes supplemental information which may be optional to your particular application of the **H & D Base** program.

The FORTH Language

H & D Base has been written with the **H & D Forth** programming language. A total of over 300 FORTH commands are included in *this* system, approximately 75 of which execute functions particular to the database. The remainder of the commands can be used by programmers familiar with FORTH to accomplish a myriad of supplementary functions such as accessing the "GEM" interface of the Atari (windows, menus, dialog boxes, etc.).

Important Note: You **must** purchase the "520ST Developer's Kit" from Atari in order to understand how to apply the supplemental "GEM" commands to the system. Without the "Developer's Kit," the task is virtually impossible.

Because these additional FORTH commands have no bearing on the majority of **H & D Base** owners, they are *not* documented in this manual. They are, however, thoroughly reviewed in a text file on the **H & D Base** Program Disk.

Printing the FORTH Manual

The name of the file which documents all commands not covered in this manual is "HDFORTH". If you would like to either review or print that documentation, follow these steps carefully:

1.) Load the TOS Operating System (if it is not in ROM).

2.) Insert your **H & D Base** Program Disk in a drive and "double click" on that drive's icon. A directory will appear.

3.) "Double click" on the file named "HDFORTH". A dialog box with the words "Show", "Print", and "Cancel" prompts will appear.

4.) If you would like to view the instructions on your computer's screen (only), "Click" on the box which says "Show".

5.) If you would like to print a copy of the instructions, "Click" on the print" box.

Typographic Conventions

In this manual, there are a number of commands to which qualifying statements or expressions may be added. For example, the simple **H & D Base** command "DISPLAY" might look like this:

```
DISPLAY [scope] [FOR {exp}] [OFF]
```

It is important for you to understand the difference between upper and lower case entries, which qualifiers you *can* add to certain commands, and the form the qualifiers *must* be in for them to work properly.

1.) **Uppercase Entries**
Anything which appears in uppercase is a **H & D Base** command or prompt.

2.) **Lowercase Entries**
Anything which appears in lowercase is something that *you* type in.

3.) **Entries in Square Brackets** [........]
Anything which appears in square brackets is a part of the command which is *optional*.

4.) **Entries in Braces** {........}
Anything which appears in braces is a part of the command which is to be filled in with *real information*. They are also used to bracket field names and file names.

5.) {**CR**}
Anytime this symbol appears, it means to press the carriage return key on the keyboard ("Return").

Using the Program

The COMMAND LEVEL

The words you will be using to create and manipulate the information you type into **H & D Base** will be entered in what is referred to as the "COMMAND LEVEL." Most of the words, or "COMMANDS" will remove you from the COMMAND LEVEL allowing you to accomplish a certain task or purpose in relation to a particular database. Once that task has been accomplished, the program will return you to the COMMAND LEVEL.

The "ok" Prompt

H & D Base uses an "ok" prompt to tell you that you are at the COMMAND LEVEL. It appears *at the end of the line prior to the line where the cursor is resting*. When you first enter the program, it will be on a line all by itself.

Command Syntax

You can enter a **H & D Base** command anytime the "ok" prompt is resident. If the COMMAND (or COMMANDS) you have entered adhere to all of the proper **H & D Base** syntax, an "ok" prompt will appear after that line of code when you press {CR}.

If a question mark appears in place of the "ok" prompt when you press {CR}, it means that the command was entered incorrectly and must be re-entered.

In some instances, "Error Messages" will appear giving you some indication of the nature of the problem the system encountered.

Command Entry Short-Cut

You may enter just *the first first four letters* of a COMMAND if you so choose, as long as that COMMAND is *not* the first word on a line. For example:

"DISPLAY STRUCTURE" is the same as "DISPLAY STRU"

This form of the command is *not* acceptable:

"DISP STRU"

COMMANDS which have been abbreviated properly will be executed as if you had typed-in entire words. Note: You may type in more than four letters, but it you do, they must all be correct.

The PROGRAM LEVEL

In working with **H & D Base** at the COMMAND LEVEL, you are allowed to enter and execute just one COMMAND at a time. Additionally, there is no way to keep track of and store the order of the COMMANDS that were entered.

You may do all of these things at the PROGRAM LEVEL of **H & D Base.** While at this level you can build complex "programs" which are nothing more than a series of COMMANDS which have been placed in a particular order and stored in a COMMAND FILE. That file may be run over and over again, accomplishing the same task with each run.

Help!

H & D Base includes an on-screen HELP system which you can access at any time while using the program. The HELP system includes a brief overview of each command and the proper syntax you must enter to activate that command.

To get HELP regarding a specific command while at the COMMAND LEVEL of the program, enter the word "HELP" followed by a blank space and the name of the command you need HELP on. For example, you can get HELP with the "LIST" command, by pressing:

HELP LIST {CR}

When you press {CR}, the information you requested will appear on the screen. Press the "Escape Key" {ESC} to return to the program.

Using the Program, cont.

Note: The file "HDBASE.HLP" must reside on the disk in the default drive or the HELP command will not work.

The CONTROL Key

Many of the functions that **H & D Base** performs are done so through the use of the "Control Key" which is located on the left of the keyboard, second row from the bottom. *The "Control Key" will always be used in association with another key on the board.*

When a command like this appears in the manual:

{CTRL-S}

it means that you should *hold down* the "Control" key (as if it were a "Shift" key) while pressing the other key in the sequence (in this case, an "S").

Cursor Movement Keys

When using certain **H & D Base** functions, you will have frequent cause to move the cursor within the confines of a particular record. The keys which control the basic movements of the cursor are:

1.) {UP ARROW} or {CTRL-E}
Moves the cursor one field up

2.) {DOWN ARROW} or {CTRL-X}
Moves the cursor one field down

3.) {LEFT ARROW} or {CTRL-S}
Moves the cursor one character to the left

4.) {RIGHT ARROW} or {CTRL-D}
Moves the cursor one character to the right

5.) {HOME}
Moves the cursor to the first character in the record

The ESCAPE Key

The "Escape" key {ESC} is located in the upper left corner of the Atari keyboard. It is used in **H & D Base** to terminate the execution of a command (or commands) in an "emergency" situation.

Important Note: The {ESC} key should be pressed during operations which involve "read" operations only (such as DISPLAY and LIST). If you press the {ESC} key during the execution of any command involving a "write" operation (such as PACK, APPEND FROM, JOIN, and UPDATE), errors in data transfer and/or storage will undoubtedly occur.

Chapter 2
STARTING UP

Booting the System

Turn the components of your computer system "ON" and insert the TOS "System Disk" (the one that came with your computer) into your disk drive (drive "A" if you have two). Make sure the "Write Protect Tab" in the upper right corner of the disk is in the "protected" position (open or up) when you insert it. You should be able to see through the hole.

The disk will spin for a number of seconds followed by the appearance of the Atari DESKTOP on the screen.

Note: If your machine contains the TOS Operating System in ROM, the Atari DESKTOP will appear automatically when you turn the computer "ON." There is no need to load the operating system from a disk.

Copying the Program Disk

Before doing *anything* with **H & D Base**, we strongly encourage you to make a copy of the Program Disk. The process involves "Formatting a Blank Disk" and "Copying the Program".

Formatting a Blank Disk

Remove the TOS "System Disk" from your drive. Insert a *blank* disk in its place (if you have two disk drives, put the blank in drive "B"). Make sure the "Write Protect Tab" in the upper right corner of the disk is in the "unprotected" position (closed or down). You should *not* be able to see through the hole.

Format the blank disk for use with your system by moving the cursor to the icon for drive "B" and pressing the *left* button on your mouse. The icon will turn black when it is selected.

Now move the mouse pointer to the Menu Bar. The "FILE" menu will drop down. Select the "FORMAT" item from the "FILE" menu by clicking on it with the left button on your mouse.

The system will paint a dialog box warning you that it is going to overwrite the disk in drive "B". *Make sure it says "Drive B"*! Click on the "OK". The dialog box for formatting will appear. Set the FORMAT to SINGLE SIDED and click the FORMAT box. When it is done, click on "Exit".

The Copy Process

Insert the **H & D Base** Program Disk in drive "A", and a blank, formatted disk in drive "B". The "Write Protect Tab" on the **H & D Base** Program Disk should be in the "Protected" position (open or up), while the same tab should be in the "Unprotected" (closed or down) position on the blank disk.

From the Atari's DESKTOP, select the disk "A" icon with your mouse and "drag" it to the disk "B" icon. A "COPY" dialog box will appear. Click the "COPY" answer button.The disk will now begin to copy. If you have only one disk drive, you will be asked to switch disks two or three times during the process. The original **H & D Base** Program Disk is your "Source" or "A" disk, and the copy is your "Destination" or "B" disk.

When the copying process is completed, you will return to the COPY dialog box. Click on the "EXIT" button, and you will be returned to the DESKTOP. Remove the original copy of **H & D Base** from your drive and *place it in a secure storage area for safe keeping*.

Loading the Program

Load the TOS Operating System in the manner described above ("Booting the System"). When the Atari DESKTOP appears on the screen, remove the "System Disk" from your drive and insert the **H & D Base** Program Disk in its place. Double "click" on the drive "A" icon and the **H & D Base** icon will appear. It is the one that says "HDBASE.PRG" under it. Double "click" on it and the program will load into the computer. The process of loading has been completed when the disk drive stops spinning and an **H & D Base** I.D. appears on the screen.

Underneath the **H & D Base** I.D. you will see a prompt which reads "ok". It indicates (as previously described) that you are at the "COMMAND LEVEL" of the **H & D Base** program.

The Data Disk

A Word of Caution

The information (data) you enter into **H & D Base** should *not* be stored on the Program Disk. This guards against accidental over-write and prevents unwarranted wear on the most important element of your system (the Program Disk).

Loading the Data Disk

Once the **H & D Base** has been loaded into your computer, remove the Program Disk from the drive and replace it with a formatted data disk. *We suggest that you do so even if you have a two drive system.*

Chapter 3
BEGINNING COMMAND TUTORIAL

Introduction

Welcome

Welcome to the *Beginning Command Tutorial*. In this section, we will be laying the foundation for your effective use of the **H & D Base** program. We assume nothing on your part except the fact that you have read through, and studied carefully the *Introduction* and *Starting Up* sections of this manual. We also *strongly* encourage you to review "Appendix A" in the back of this manual which deals with the nature of a database management system -- especially if this is your first experience with this type of program.

Starting a Session

Before you begin, turn on all the elements of your computer system and load the **H & D Base** program as instructed in the "Starting Up" section of this manual. If you have done everything correctly, an "ok" prompt will appear on the screen. It indicates that you are at the COMMAND LEVEL of the program. Before proceeding, we recommend that you remove the Program Disk and insert a data disk in the appropriate drive.

Turning On the Printer

Anytime you would like to have the program "display" its work (and yours) on your printer as well as your screen, enter:

```
SET PRINT ON {CR}
```

To turn the printer "off," enter:

```
SET PRINT OFF {CR}
```

The logic behind these commands will appear in a subsequent section of this manual.

The ERASE Command

If you would like to clear the screen of all text at any given time, enter this command:

Introduction, cont.

```
ERASE {CR}
```

All text will disappear and the cursor will reposition itself to the upper left corner of the screen.

Using Alternate Drives

Whenever you enter the name of a file to be used in relation to a certain command, **H & B Base** assumes that the file resides on the data disk in the drive that the program was loaded from. You may indicate to the program that a file resides on a disk in an alternate drive by prefacing the file name with the letter which represents that drive followed by a colon.

For example:

```
USE People                    (File located on default drive)
USE B:People                     (File located on "B" drive)
```

The "SET DEFAULT" command allows you to choose an alternate disk drive that the program will access when searching for files which appear *without* a designated drive letter preceeding them.

The QUIT Command

You are not instructed anywhere in the tutorial how to exit the **H & D Base** program. This is because we have no way of determining where your sessions will begin and end in relation to the text.

To exit the program from the COMMAND LEVEL, simply enter this command:

```
QUIT {CR}
```

You will be returned to the Atari's DESKTOP.

Creating a Database

The CREATE Command

For the purpose of demonstration, our first task will be to create a simple database. While at the program's COMMAND LEVEL, enter the following:

```
CREATE {CR}
```

It can be in either upper or lower case, it makes no difference. After pressing {CR}, these words will appear:

```
ENTER NEW FILE NAME:
```

File Name Parameters

You may enter any name you want, provided that it falls within the following parameters:

1.) It must start with a letter (not a number)

2.) It can be up to 8 characters long

3.) It cannot have any colons and/or spaces

Let's name our demonstration database "People". It makes no difference if it is entered in upper or lower case, but we suggest that you do a combination of both (upper and lower).

To the prompt "ENTER NEW FILE NAME", type:

```
People {CR}
```

".DAT" Files

When you have done so, **H & D Base** will create a file named "People.DAT". The trailer which follows "PEOPLE" (.DAT) stands for "database file," and will appear after the name of every database you create.

Creating a Database, cont.

Entering a Database Structure

The program will now ask you to enter the "Structure" of your new file via a screen which looks like this:

```
ENTER THE FIELDS FOR THE NEW DATABASE:
NAME - NAME OF FIELD, MAXIMUM LENGTH = 10
TYPE - CHARACTER (C), NUMERIC (N), OR LOGICAL (L)
LENGTH - MAXIMUM LENGTH = 254
DECIMALS - NUMBER OF DIGITS TO RIGHT OF DECIMAL POINT
FIELD NO.   NAME,TYPE,LENGTH,NO. OF DECIMAL DIGITS
01
```

You are being prompted to enter information regarding your new database. The information must be entered as detailed on the screen; e.g., the NAME followed immediately by a comma, followed immediately with the TYPE, followed immediately by a comma...and so on.

Database Structure Parameters

The parameters for database structure are as follows:

1.) The NAME of each field in the database:
Enter a name to describe the category of information which the field is to contain. Parameters for field names are as follows:
A.) 1-10 characters
B.) Upper and/or lower case
C.) Must start with a Letter
D.) Cannot contain spaces
E.) Can contain digits and embedded colons

2.) The TYPE of data to be stored in each field:
Enter one of the following three characters:
A.) "C" = Character Field
A field containing letters, numbers, or symbols, or a combination of letters, numbers, and symbols
B.) "N" = Numeric Field
A field containing entries which are all numeric
C.) "L" = Logical Field
A field containing only "Y" for "YES", "N" for "NO, "T" for "TRUE", or "F" for "FALSE".

3.) The LENGTH of each field (number of characters):
Enter any number of characters from 1 to 254.
Note: If the field is numeric, and decimal places are specified, the decimal point will occupy one character position.

4.) The number DECIMAL PLACES in each numeric field: Enter any number of characters (from 1 to 8) to the *right* of the decimal point (not including the decimal point itself).

Note: You need to specify the number of DECIMAL PLACES *only on fields which are numeric.*

Our sample "People" database will consist of the following six fields:

"Name"	- C -	30 Characters -	0 Decimals
"Address"	- C -	30 Characters -	0 Decimals
"City"	- C -	20 Characters -	0 Decimals
"State"	- C -	2 Characters -	0 Decimals
"Zip"	- C -	5 Characters -	0 Decimals
"Amount"	- N -	7 Characters -	2 Decimals

Enter them at this time. When you are to a point where you would begin to enter information in field #7, simply press {CR}. This signals the program that there will be only six fields in your database.

When you are finished, the screen should look like this:

```
ENTER THE FIELDS FOR THE NEW DATABASE:
NAME - NAME OF FIELD, MAXIMUM LENGTH = 10
TYPE - CHARACTER (C), NUMERIC (N), OR LOGICAL (L)
LENGTH - MAXIMUM LENGTH = 254
DECIMALS - NUMBER OF DIGITS TO RIGHT OF DECIMAL POINT
FIELD NO.   NAME, TYPE, LENGTH, NO. OF DECIMAL DIGITS
01  Name,C,30
02  Address,C,30
03  City,C,20
04  State,C,2
05  Zip,C,5
06  Amount,N,7,2
07  {CR}
```

Creating a Database, cont.

Another Form of the CREATE Command

There is one other form of the CREATE command which is available to you in **H & D Base**.

CREATE FROM - This command creates a new file whose structure is determined by the data in the records of the old file.

For information on its correct use, consult the "Reference" section of this manual.

Modifying the Structure of a Database

If, before you enter actual data into a database, you would like to change the structure of the database, there is a command which will do so:

```
MODIFY STRUCTURE
```

One word of caution: If you modify the structure of a database which has data in its records, all of that data will be erased.

For information on the correct use of the MODIFY STRUCTURE command, consult the *Advanced Command Tutorial* or *Reference* sections of this manual.

Using a Database

The USE Command

Once a database has been created, its structure is stored on the data disk along with all of the other files you have established. Therefore, before you can add or manipulate the records of a particular database, you must first tell the program which file you want to work with. The USE command fills this purpose.

When you want to select a database file for use, make sure you are at the COMMAND LEVEL then type:

```
USE {filename}
```

You do *not* have to include the ".DAT" trailer. (If you don't, the program will insert it for you.)

The program will pause for an instant, and the "OK" prompt will reappear. Its appearance signifies that the file has been successfully "opened," and any command entered from this point on will be executed in direct relation to the file you specified. It will stay "open" until you USE another file, CLEAR memory (covered later), or QUIT the program.

Enter the following at this time:

```
USE People {CR}
```

This will allow us to access the demonstration file we created in the first portion of this tutorial.

Entering Information Into a Database

Once you have completed the task of creating a database, you can begin entering information into it. This is the process of "APPENDING," and it is a simple concept to understand.

You cannot add information to a database if you have not told the program which database it is you want to work with. You must USE a database before you can APPEND to it.

The APPEND Command

We are currently using our sample database named "People." To add information to it, type:

APPEND {CR}

This screen will appear:

```
RECORD 00001
NAME      :                                      :
ADDRESS   :                                      :
CITY      :                      :
STATE     :  :
ZIP       :     :
AMOUNT    :        :
```

The cursor is positioned on the first character of the first field of the next available record in the file. Because there are no records in our sample file, this is the first record.

The length of each field is defined by colons marking the beginning and end of the field.

Entering Information Into a Database, cont.

Entering Information

To enter data into a particular field, simply type-in the information and press {CR}. If you attempt to enter more characters than the field will hold, an automatic {CR} will be issued by the program (it will also "beep"). To leave a field blank, press only {CR}.

Because the first five fields were defined in the CREATE process as "Character" fields, you may enter any character in them that you desire. Attempt to enter anything other than a number in the "AMOUNT" field, however, and you will be prevented from doing so. This is because the field was defined as a "Numeric" field when it was first created. Notice that all entries in this type of field are "right justified" upon pressing {CR}.

Cursor Control Commands

A number of cursor control commands have been provided for use while in the APPEND mode. They are detailed in the *Introduction* to this manual.

Miscellaneous Key and Control Functions

There are a number of APPEND functions which are activated through the use of "Control Key" sequences. They are as follows:

1.) {CTRL-Y}
Deletes from cursor position to end of field

2.) {CTRL-G} or {DELETE}
Delete character under cursor

3.) {BACKSPACE}
Delete character to left of cursor

4.) {CTRL-W}
Exit - Save

5.) {CTRL-Q}
Exit - No save

Entering Information Into a Database, cont.

Note: Some of the control functions you will learn about in other sections of this manual will *not* work in the APPEND mode. The list above is the *complete* list of applicable commands for the APPEND mode.

Enter the following information into our "People" database at this time:

Smith, Paul	5487 Oak St.	
	San Diego, CA 92376	$267.92
Jones, Alice	9227 E. Sample #108	
	Miami, FL 39857	$ 67.98
Zachry, Mike	746 Lover's Lane	
	Denver, CO 85678	$957.00
Dow, Tony	1987-B E. 19th.	
	New York, NY 01004	$ 4.59
Clark, Bill	657 S. Main	
	Portland, ME 00648	$ 34.95

If you make a mistake, use the cursor control features mentioned above and edit what you can, but don't worry about getting everything in exactly right. You can work on that later in the actual EDIT section.

Moving to the Next Record

Once you have pushed {CR} from the *last* field in the record (in this case the "AMOUNT" field), the program will automatically go on to the next record. Once you have advanced to the next record in the APPEND mode, you cannot go back to a previous record.

Exiting the APPEND Mode

You can exit the APPEND mode in three ways:

1.) Press {CR} while the cursor rests on the first (non)character of a completely blank record

Entering Information Into a Database, cont.

2.) Press {CTRL-W} -- (Any information in that record will be saved)

3.) Press {CTRL-Q} -- (Any information in that record will *not* be saved)

When you are finished entering the information for our sample database, "exit" the APPEND mode in one of the manners described above.

Another Form of the APPEND Command

There is one other form of the APPEND command which is available to you in **H & D Base**.

APPEND FROM - This command takes the information from another database and merges into that of the one in use.

For information on its correct usage, consult the *Advanced Command Tutorial* or *Reference* sections of this manual.

Inserting Records into a Database

Records can be inserted into a specific location in a database file (instead of being added to the end) using the INSERT command. This command is useful for keeping a SORTED database in order.

For information on the INSERT command, consult the "Reference" section of the manual.

Changing Information In a Database

Correcting or updating the information in the records of a database is simple with **H & D Base**. The two commands reviewed in this section, EDIT and BROWSE, are used precisely for this purpose.

Record Numbers

Every record in a file has a "Record Number" which is associated with it. The number corresponds to the order in which the record was entered into the database (unless the file has been SORTED).

The very first name and address you entered into our sample database ("People") became "Record #00001". You entered five names, so there are five records in the file numbered consecutively from #00001 to #00005.
While at the COMMAND LEVEL, type-in the following:

```
USE People {CR}
```

(Note: If you are continuing from an earlier portion of this tutorial, it is actually not necessary to enter the USE command with each new phase.)

The EDIT Command

The first of the two commands we will review in this section is the EDIT command. There are two ways you can use the command to access the precise record of the database you want to EDIT:

1.) Enter just the following:

```
EDIT {CR}.
```

In response, the program will place the current record on the screen.

2.) Enter the EDIT command followed by the number of the record you want to EDIT:

EDIT {*Record Number*} {CR}

In response, **H & D Base** will place the desired record on the screen.

For demonstration purposes, enter the following:

EDIT {CR}

The first record of our "People" database should appear on the screen (it was the "current" record).

```
RECORD 00001
NAME      :Smith, Paul                    :
ADDRESS   :5487 Oak St.                   :
CITY      :San Diego             :
STATE     :CA:
ZIP       :92376:
AMOUNT    : 267.92:
```

Cursor Control Commands

A number of cursor control commands have been provided for use while in the EDIT mode. They are detailed in the *Introduction* section of this manual.

Miscellaneous Key and Control Functions

There are a number of EDIT functions which are activated through the use of "Control" key sequences. They are as follows:

1.) {CTRL-T}
Delete from cursor position to end of field

2.) {CTRL-Y}
Delete line

Changing Information In a Database, cont.

3.) {CTRL-G} or {DELETE}
Delete character under cursor

4.) {BACKSPACE}
Delete character to left of cursor

5.) {CTRL-U}
Delete record toggle

Pressing {CTRL-U} *marks* a record for eventual deletion (a prompt appears at the top of the screen). The record is *not* deleted at the time the command is entered. For further information, consult the section on "Packing a File" at the end of this *Beginning Tutorial*.

6.) {CTRL-A} or {CTRL-R}
Save record - Back up to previous record

7.) {CTRL-F} or {CTRL-C}
Save record - Advance to next record

8.) {CTRL-W}
Exit - Save changes

9.) {CTRL-Q}
Exit - Do *not* save changes

Special Note: Atari has chosen to designate {CTRL-C} as the code for exiting from a program and returning to the DESKTOP. It is "hard wired," but can be overriden (to a certain degree) by software programmers. If you press {CTRL-C} two or more times *in rapid succession*, you will automatically exit **H & B Base** and return to the DESKTOP. We suggest, therefore, that you get in the habit of using either {CTRL-F} or the {DOWN ARROW}.

You might want to spend a few minutes working through our sample database using the control commands outlined above. If you make any changes in the data of a record, be sure to change it back before moving to another. When you have satisfied your curiosity, press {CTRL-Q} to exit the EDIT mode.

The BROWSE Command

The second of the two commands we will review in this section is the BROWSE command. As opposed to the EDIT command which allows you to EDIT but one record at a time, the BROWSE command places a number of records (or parts of records) on the screen still allowing you to edit any one of them at will.

There are two ways you can use the BROWSE command:

1.) Enter just the following:

```
BROWSE {CR}
```

Upon pressing {CR}, the first 19 records of the file will appear on the screen. Each will occupy one line.

If the number of characters in all fields (combined) is greater than 80, part of the record will not be displayed on the screen. It is beyond the right margin. You may access the information with the commands outlined below.

2.) If you would like to edit only a certain field or fields of your database, you can enter the command:

```
BROWSE FIELDS {Name of field/s}
```

If you are entering more than one field, separate each name with a comma.

As an example, let's BROWSE the "Name", "City" and "Amount" fields of our "People" database. Enter the following from the program's COMMAND LEVEL:

```
BROWSE FIELDS Name, City, Amount {CR}
```

When you press {CR} the program will list only the field or fields you specified in the command.

Changing Information In a Database, cont.

Cursor Control and Function Commands

Except for {CTRL-T} and {CTRL-Y}, you may use the same cursor movement commands and function commands that you did in the EDIT mode. In addition, the following three commands are activated:

1.) {CTRL-B}
Scroll the screen one *field* to the right

2.) {CTRL-Z}
Scroll the screen one *field* to the left

3.) {CTRL-N}
Insert a line (for a new record)

Spend a few minutes now using the BROWSE function. If you make any changes, be sure to change them back. When you are finished, press {CRTL-Q} to return to the program's COMMAND LEVEL.

Reviewing a Database

The ultimate goal of all database operation is to display the data in a file in any number of different forms -- upon demand. Because that is so, the DISPLAY and LIST commands are the foundations of the entire **H & D Base** system.

The Record Pointer

When, from the COMMAND LEVEL, you specify a database for manipulation with the USE command, a "pointer" (arrow) is placed in memory on record #00001. In most cases, this "pointer" stays at the "top" of the file or is reset to that point when it exits a given mode. (You will learn how to move the "Pointer" later in this section.)

The DISPLAY Command

Take a look at our sample database and see which record the "pointer" is on. From the COMMAND LEVEL enter this new command:

```
DISPLAY {CR}
```

This record will appear on the screen:

```
00001   Smith, Paul   5487 Oak St.
    San Diego   CA   92376   $267.92
```

It is record #00001, the first record in the file. Because you are using the DISPLAY command, you can do nothing more than just look at it. You are returned to the COMMAND LEVEL automatically.

Expanding Commands

One of the most powerful features of **H & D Base** is the ability it gives you as a user to add further definition to commands such as DISPLAY. Such definition can take a number of forms, two of

Reviewing a Database, cont.

which are covered below. Perhaps the most powerful of all expansion tools is the ability the program has to evaluate "expressions" ({exp}). You will review expressions in the *Intermediate Command Tutorial* of this manual.

Our purpose here is to give you *just an overview* of how commands can potentially be expanded. We will be looking at two of the more common forms as they relate to a command like DISPLAY.

1.) DISPLAY {field}

You may perform an action in relation only to a certain field (or fields) of a database with a qualifier like "{field}".

Our sample database named "People" has 6 fields in each record. In order to DISPLAY only the "Name" field we would enter:

```
DISPLAY Name
```

A command DISPLAYING three fields might read:

```
DISPLAY Name,City,Zip
```

2.) DISPLAY {scope}

The word {scope} is used to tell the program the range of application for the COMMAND it is used in conjunction with. It can take three forms:

ALL
All records in the file

NEXT *n*
Next *n* records including the current record

RECORD *n*
Only record *n*

Used in relation to the DISPLAY command, the correct form of each command would read:

ALL - (Ex: DISPLAY ALL)
Displays all records in the file

NEXT *n* - (Ex: DISPLAY NEXT 5)
Displays the next 5 records including the current record

RECORD *n* - (Ex: DISPLAY RECORD 3)
Displays record #00003

Let's experiment for a moment with just one of the DISPLAY commands reviewed above. Enter the following:

DISPLAY ALL {CR}

Instead of displaying just one record, all five will appear on the screen.

```
00001 Smith, Paul                 5487 Oak St.
San Diego                            CA 92376 $267.92
00002 Jones, Alice                9227 E. Sample #108
Miami                                FL 39857 $  67.98
00003 Zachry, Mike                746 Lover's Lane
Denver                               CO 85678 $957.00
00004 Dow, Tony                   1987-B E. 19th.
New York                             NY 01004 $  4.59
00005 Clark, Bill                 657 S. Main
Portland                             ME 00648 $  34.95 ok
```

With a file which has a large number of records in it, the first 15 records of the file will appear when the DISPLAY ALL command is entered. In order to DISPLAY the next 15 records of the database, you would press *any* key on the keyboard. You would break out of the DISPLAY mode at any time by pressing the {ESC} key.

Reviewing a Database, cont.

Alternate Forms of the DISPLAY Command

In this tutorial, we have touched on just a few of the forms the DISPLAY command can take. Others include:

1.) DISPLAY STRUCTURE
Displays the structure of a database

2.) DISPLAY MEMORY
Displays the contents of all memory variables

3.) DISPLAY FILES
Displays the disk directory

4.) DISPLAY STATUS
Displays current files open, index files, keys, and SET parameters

5.) DISPLAY COMMAND {File}
Displays a command file

It will be very important for you to carefully study the "Reference" section of this manual in order make full use of the powerful commands we've briefly reviewed in this DISPLAY section.

The LIST Command

The LIST command is almost identical to the DISPLAY command with one important difference. If you enter:

```
LIST
```

without any qualifying statements after it, the *entire* file will be displayed (as opposed to one particular record with the DISPLAY command). In addition, the file will be listed from beginning to end *without stopping*.

You may manually pause the listing by entering {CTRL-S}. To continue after doing so, press {CTRL-Q}. To break out of the listing all together, press the {ESC} key.

Other than that, you will have no trouble in substituting the word LIST for the word DISPLAY in any of your statements.

You may want to take a few minutes at this time to return to the section which covers the DISPLAY command and rework the examples there using the word "LIST" instead of "DISPLAY" in the statements.

The GOTO and GO Commands

You can place the "pointer" on any record you want by entering one of these commands:

1.) GOTO [RECORD] #
   Places the "pointer" on the specified record

2.) GOTO TOP
   Places the "pointer" on the first record of the file

3.) GOTO BOTTOM
   Places the "pointer" on the last record of the file

Note: The word "GO" can be substituted for "GOTO" in the command if you so desire.

The SKIP Command

If you would like to move the "pointer" to the next record in the file, type:

```
SKIP {CR}
```

You may SKIP ahead (or back) more than just one record. Use this command:

SKIP +*n* or SKIP -*n*

There *must* be a space after the the word "SKIP".

To move the pointer from record #00001 to record #00005 type:

Reviewing a Database, cont.

```
SKIP +4 {CR}
```

To move the pointer back to record #3, type:

```
SKIP -2 {CR}
```

Now experiment with moving the "pointer" about the file with GOTO and SKIP statements. To determine your position in the file, DISPLAY the record after each move.

Purging a Database of Unwanted Records

The DELETE Command

We encourage you to regularly purge outdated records from your databases. This will cause the program to work faster and allow your storage device to handle a greater number of current records (among other things).

There are three ways that records can be DELETED from a file. Two of them are covered here:

1.) Press {CTRL-U} while in the EDIT or BROWSE modes

To delete the record upon which the cursor is resting while in the EDIT or BROWSE modes, simply press {CTRL-U}. A "DELETED" prompt will appear at top of the screen.

2.) Enter this command while at the COMMAND LEVEL:

```
DELETE {scope}
```

Ex: DELETE ALL
Deletes all records

Ex: DELETE RECORD 3
Deletes record #00003

Ex: DELETE NEXT 5
Deletes next 5 records beginning with current record

The third form of the DELETE command, which is:

```
DELETE FOR {exp}
```

allows you to DELETE just the records satisfying a certain term or condition. It uses the "FOR {exp}" statement covered in the *Intermediate Command Tutorial*.

Purging a Database of Unwanted Records, cont.

Marking Records for Deletion

When the DELETE command is entered, the specified file (or files) is not actually removed from the database. It is *marked for removal* with an asterisk by its record number.

Lets DELETE a record at this time. From the COMMAND LEVEL, enter:

```
DELETE RECORD 1 {CR}
```

With that accomplished, DISPLAY the records in the file with this command:

```
DISPLAY ALL {CR}
```

All five records will be displayed on the screen. Note that record #00001 has been marked for deletion with an asterisk opposite its number.

Special note: Records marked for deletion *can* be displayed and edited, but they *cannot* be copied, appended, or sorted.

The RECALL Command

There are three ways that you can "undelete" or RECALL records which have been marked for deletion:

1.) Press {CTRL-U} while in the EDIT or BROWSE modes

To RECALL the deleted record upon which the cursor is resting (while in the EDIT or BROWSE modes), simply press {CTRL-U}. The "DELETED" prompt will disappear from the top of the screen.

2.) Enter this command while at the COMMAND LEVEL:

```
RECALL {scope}
```

Ex: RECALL ALL
Undeletes all records which have been deleted

Purging a Database of Unwanted Records, cont.

Ex: RECALL RECORD 3
Undeletes deleted record #00003

Ex: RECALL NEXT 5
Undeletes next 5 records beginning with current record (if any or all of them had been previously deleted)

The third form of the RECALL command, which is:

```
RECALL FOR {exp}
```

allows you to RECALL just the records satisfying a certain term or condition. It uses the {exp} statement covered at the beginning of the *Advanced Command Tutorial*.

When the RECALL command is entered in relation to a previously deleted record or set of records, the asterisk which appeared by that record's number is removed.

We have one deleted record in our file -- record #00001. Let's RECALL it at this time. From the COMMAND LEVEL, enter:

```
RECALL RECORD 1 {CR}
```

With that accomplished, DISPLAY the records in the file with this command:

```
DISPLAY ALL {CR}
```

All five records will be displayed on the screen. Note that the asterisk opposite record #00001, which had marked it for removal, has disappeared.

The PACK Command

The command which physically removes from the file all records marked with an asterisk for deletion is:

```
PACK
```

Purging a Database of Unwanted Records, cont.

Besides completely eliminating records marked for deletion, the PACK command subtracts one digit from the record number of any record whose number is greater than that of a record which has been deleted.

After PACKING a file, information from a record which had previously been deleted cannot be retrieved *in any manner.*

Important Note: The PACK function is one of the most tenuous that the system performs. We strongly suggest, therefore, that you make a "Back-up" copy of a database file before you PACK it. In addition, *never* press the "Escape" key {ESC} during a PACK; it will most assuredly cause an error in data transfer.

Interacting With the System

Before concluding the *Beginning Command Tutorial*, we must cover one last item which will be important for you to understand before starting the next section. It involves the correct procedure for interacting directly with the system.

The ? Command

H & D Base provides one simple command which allows you to access the mathematical and memory functions of the machine. This command is a "question mark" (?).

To use **H & D Base** as if it was a calculator, enter a question mark followed by a quantity or mathematical function you want evaluated. Upon pressing {CR}, the answer appears on the *next* line. Enter two question marks followed by a function to be evaluated and the answer will appear on the *same* line.

The ? Command: Math

Give it a try. Enter the following from the COMMAND LEVEL:

```
? 1 + 1 {CR}
? 10-6 {CR}
?? 5*5 {CR}
?? 66/6 {CR}
```

Note: If you are unfamiliar with the meaning of the mathematical symbols used in the statements above, consult the *Intermediate Command Tutorial* or "Reference" sections of this manual.

When you have finished, the screen should look like this:

```
? 1 + 1
2 ok
? 10-6
4 ok
?? 5*5 25 ok
?? 66/6 11 ok
```

Interacting With the System, cont.

The question mark command displays answers to a mathematical operation *to the same number of decimal places as the maximum in the numbers entered* (limit = 8).

To demonstrate the fact, enter the following:

```
? 99/7 {CR}
? 99.00/7 {CR}
? 99/7.0000 {CR}
```

Now check to see if the screen doesn't look like this:

```
? 99/5
? 14 ok
? 99.00/7
? 14.14 ok
? 99/7.0000
? 14.1429 ok
```

The question mark (?) command can also be used to display the contents of certain fields of a database. To demonstrate, make sure the data disk with our sample database "People" is in the drive and enter the following:

```
USE People {CR}
GOTO 3 {CR}
? Name {CR}
GOTO BOTTOM {CR}
? Name,City,Amount {CR}
```

When you finish, the screen will look like this:

```
USE People ok
GOTO 3 ok
? Name
Zachry, Mike ok
GOTO BOTTOM ok
? Name,City,Amount
Clark, Bill                 Portland                 34.95 ok
```

Summary

This concludes the *Beginning Command Tutorial* of **H & D Base**. In it you have learned how to interact with the program using some of the most primary statements of the programming language.

In many cases, you were shown only the most basic applications of the words we covered. To make use of the full of **H & D Base** in regard to these (and other) commands, you *must* study the "Reference" section of this manual carefully.

The next section deals with "Expressions" -- the phrases which "pack the punch" of the **H & D Base** program.

Chapter

4
INTERMEDIATE COMMAND TUTORIAL

Introduction

Congratulations on having completed your review of the *Beginning Command Tutorial*. We trust that you feel as though you are well on your way to mastering the **H & D Base** programming language.

You are about to begin the *Intermediate Command Tutorial*. It is called *Intermediate* not because it follows the *Beginning Tutorial* with a logical presentation of more advanced commands, but because it is essential that we present the information provided in it before we can proceed any farther.

This section covers "Expressions" -- one of the most powerful elements of **H & D Base**. "Expressions" give you the ability to expand many of the basic commands covered in this *Tutorial* and manipulate your databases in a virtually limitless number of ways.

Unfortunately, there is no way for us to document completely all the forms each of the commands can take. Experimentation will, therefore, be your best teacher. The process can be as complicated as you make it. The deeper you dig, the harder it becomes. Good luck!

Before beginning this section, it is necessary to introduce the statement to which they are most closely tied. That statement is "FOR".

The FOR Statement

The FOR statement is used to determine which records of a file satisfy the terms of a qualifying "expression" ({exp}). For example, using the DISPLAY command you could take a look at only the records of our sample "People" database with "CA" in the "State" field, or all the records with a number in the "Amount" field greater than "100".

The correct form of the statement when applied to any given command is:

```
{COMMAND} [FOR {expression}]
```

Expressions

The real power of any given "FOR" statement is in the "Expression" which is appended to it. When you have mastered their application, you have passed a major hurdle in learning **H & D Base**.

Expression Parameters

First the parameters for entering such statements:

1.) They can be entered in upper and lower case letters.

2.) They can be up to 254 characters long.

3.) At the PROGRAM LEVEL of the program, if a command is longer than 80 characters, place a "tilde" sign (~) at the end of the last full word of a line, press {CR} and continue the command on the next line. The program will read it all as one command. The tilde is not necessary while at the COMMAND LEVEL of **H & D Base**.

Note: Those of you who have worked extensively with other database languages know that a semi-colon is normally used in this situation. The nature of FORTH does, unfortunately, necessitate the change.

Elements of Expressions

Elements which may be included in an "Expression" are:

1.) CONSTANTS
- A.) Numerical Constant
- B.) Character String Constant
- C.) Logical Constant

2.) VARIABLES
- A.) Data Field Variable
- B.) Memory Variable

3.) OPERATORS
- A.) Arithmetic
- B.) Relational
- C.) Logical
- D.) String

4.) FUNCTIONS

We will cover each of the four in the remainder of this *Intermediate Tutorial.*

Constants

A "Constant" is a data item which always has the same *literal* value -- regardless of where it appears. It *is* exactly what it says. A constant can take three forms: a numerical constant, a character string constant, or a logical constant.

Numeric Constants

A "numeric constant" is no more than a single number (or numbers) such as "1" or "12345".

For purposes of demonstration, enter the following from the COMMAND LEVEL of **H & D Base**:

```
? 12345 {CR}
```

The numbers "12345" are returned to the screen when you press {CR}. The computer determined that they were "numeric constants" and returned their literal value to the screen.

Now enter:

```
? ABCDE {CR}
```

What do you get? A "question mark" (?), indicating that a "syntax error" has occurred. Letters by themselves are *not* constants and cannot be interpreted as such by the computer.

String Constants

A "string constant" is *any printable character and/or spaces* which appear between a set of matching quotes (single or double) as in "ABC D/*X + 12 34%$#@". Note: The ampersand (&) must receive special handling if it appears in such a statement. (Consult the *Reference* section.)

Enter the following:

```
? "ABCDE" {CR}
```

Instead of a "Syntax Error," it returns "ABCDE." This is because it was in quotes and the computer interpreted it as a "String Constant."

Logical Constants A "logical constant" is a value representing "True" with "T", "t", "Y", or "y", and "False" with "F", "f", "N", or "n".

Enter:

```
? T {CR}
```

According to what we've learned so far, a "syntax error" should be issued. It is not. In its place, a ".T." appears. This is because any of the eight characters listed above, when entered without quotes, is interpreted by the computer as a "logical constant" and a "true" (.T.) or "false" (.F.) value is issued.

Variables

A "Variable" is any data item whose value may change. There are two types of variables: "Data Field Variables" (representing information contained in the field of a record), and "Memory Variables" (representing information which is stored in special sections of the computer's memory).

Variables may look exactly like constants, but differ in that they are not "literal" or "permanent" but are instead "representative" and "transitory".

Data Field Variables

Each of the names given to the fields in the databases you design is a "Data Field Variable." That's because it *represents* a data value which may change.

If you recall, there are six fields in our sample database called "People". They are "Name", "Address", "City", "State", "Zip", and "Amount". Each is a "Data Field Variable."

Take a look at the contents of one of the fields by entering the following from the COMMAND LEVEL of the program:

```
USE People {CR}
? Name {CR}
```

The computer returns the name "Smith, Paul". It is the information from the NAME field of the record #00001 of the database. Why record #00001? Because that's the record the "pointer" was on when you entered the command. Note that the word "Name" is a variable; it *represents* a value which, at some time, may change.

If you had entered the command "? Name" without first specifying a database to USE, the error prompt "Invalid Name or String" would have been returned.

Memory Variables

Chester and I reserved an area of the computer's memory for you to place temporary information in while using the program. We divided that memory into 150 boxes of up to 254 characters each (maximum of 3000 characters total for *all* variables), and allow you to place a name on the outside of each box. These names are "Memory Variables."

Naming Memory Variables

The rules for naming "Memory Variables" are the same as those for naming fields of a database; e.g., they must start with an alphabetic character, can be no longer than 10 characters, can have embedded colons and numbers, and may include no spaces. We suggest that you begin the name of all "memory variables" with the letter "M" as an indication of their nature.

The STORE Command

To place data into a "memory variable," you can use the STORE command. Its full form is:

```
STORE {exp} TO {memvar}
```

Enter this sequence of commands. It will demonstrate some of the functions of the STORE command:

```
STORE 5 TO Mnumber1 {CR}
STORE 10 TO Mnumber2 {CR}
STORE "This is a String Constant" TO Mstring {CR}
? Mnumber1 {CR}
? Mnumber2 {CR}
? Mnumber1 * Mnumber2 {CR}
? Mstring {CR}
```

Variables, cont.

When you finish, the screen should look like this:

```
STORE 5 TO Mnumber1 ok                                Line 1
STORE 10 TO Mnumber2 ok                               Line 2
STORE "This is a String Constant" TO Mstring ok       Line 3
? Mnumber1                                            Line 4
5 ok                                                  Line 5
? Mnumber2                                            Line 6
10 ok                                                 Line 7
? Mnumber1 * Mnumber2                                 Line 8
50 ok                                                 Line 9
? Mstring                                             Line 10
This is a String Constant ok                          Line 11
```

Line-by-line, this is what occured:

Line 1 — You stored a numeric constant (5) to a memory variable named "Mnumber1"
Line 2 — You stored a numeric constant (10) to a memory variable named "Mnumber2"
Line 3 — You stored a string constant ("This is a String Constant") to a memory variable named "Mstring"
Line 4 — You asked for the contents of "Mnumber1" to be placed on the screen
Line 5 — The program placed the contents of "Mnumber1" on the screen
Line 6 — You asked for the contents of "Mnumber2" to be placed on the screen
Line 7 — The program placed the contents of "Mnumber2" on the screen
Line 8 — You asked the program to multiply the contents of "Mnumber1" and "Mnumber2" and place the result on the screen
Line 9 — The program placed the results of the computation in Line 8 on the screen
Line 10 — You asked for the contents of "Mstring" to be placed on the screen
Line 11 — The program placed the contents of "Mstring" on the screen

You may want to take a few moments to experiment with creating "memory variables" and placing miscellaneous values into them.

The RELEASE Command

You may "dump" (or eliminate) a "memory variable" and the data in the box it represents by entering this command:

```
RELEASE {Memory Variable}
```

To clear the computer of *all* "memory variables," enter:

```
RELEASE ALL
```

When you do so, all "memory variables" are *erased*.

"Memory Variables" are also cleared with the "QUIT" and "CLEAR" commands which are covered elsewhere in this manual.

The DISPLAY MEMORY Command

Any time you would like to look at all of the boxes of memory *containing information*, you may enter this command:

```
DISPLAY MEMORY (You can also LIST MEMORY)
```

The "memory variable" names will appear on the screen along with their data type and contents. At the bottom you will be shown the number of variables (out of 64) you have used and how much of the computer's memory they occupy.

You may SAVE "memory variables" to a file for future use with the SAVE TO command. They may be retrieved with the RESTORE FROM command. Information on these commands is included in the "Reference" section of this manual.

Operators

"Operators" are manipulations that **H & D Base** can perform on your data. There are four basic types of operations:

1.) Arithmetic
2.) Relational
3.) Logical
4.) String

We will cover each briefly in this section.

Arithmetic Operators

Perhaps the most familiar of all "Operators" are those which are "Arithmetic." There are four of them:

1.)	*	:Multiply
2.)	/	:Divide
3.)	+	:Add
4.)	-	:Subtract

Parentheses ( ) are used to group a series or operations together.

Because the use of these symbols is so common, we will not spend a great deal of time at this point teaching you how they operate.

It is, however, important for you to realize the fact that manipulations are performed in a sequence of precedence which will always return the same value for phrases like:

10*10+10

Is the answer here "110" or is it "200"? It depends on what order the calculations are being performed in.

The order of precedence is:

1.) Anything in parentheses
2.) Divide and Multiply
3.) Add and Subtract

When "Arithmetic Operators" have equal precedence (as in "Divide and Multiply"), they are evaluated from left to right.

Some examples:

```
195/26*14+9    = 114     (divide, multiply, add)
195/(26*14+9)  = .5228   (multiply, add, divide)
195/26*(14+9)  = 172.5   (add, divide, multiply)
```

Relational Operators

"Relational Operators" make comparisons then generate logical results ("True" or "False"). Action is then taken based on that result.

There are seven "Relational Operators." They are:

1.)	<	:less than
2.)	>	:greater than
3.)	=	:equal to
4.)	< >	:not equal to
5.)	< =	:less than or equal to
6.)	> =	:greater than or equal to
7.)	$	:substring is within string

The use of the first six is simple and straight forward. For example, type in the following from the COMMAND LEVEL of the program:

```
USE People {CR}
LIST FOR Zip = 92376 {CR}
LIST FOR Amount > 100 {CR}
LIST FOR Zip < > 92376 {CR}
```

Operators, cont.

When you have finished, the screen should look like this:

```
USE PEOPLE ok
LIST FOR Zip = 92376
00001  Smith, Paul   5487 Oak St.
   San Diego  CA  92376  $267.92  ok
LIST FOR Amount > 100 ok
00001  Smith, Paul   5487 Oak St.
   San Diego  CA  92376  $267.92
00003  Zachry, Mike  746 Lover's Lane
   Denver     CO  85678  $957.00  ok
LIST FOR Zip < > 92376
00002  Jones, Alice  9227 E. Sample #108
   Miami      FL  39857  $ 67.98
00003  Zachry, Mike  746 Lover's Lane
   Denver     CO  85678  $957.00
00004  Dow, Tony     1987-B E. 19th.
   New York   NY  01004  $  4.59
00005  Clark, Bill   657 S. Main
   Portland   ME  00648  $ 34.95  ok
```

Take a moment to think about what occurred. When you entered a LIST command with its qualifiers, the program searched through the file to see if any record satisfied its terms. Any record that did returned a logical "True" and was, therefore, LISTED on the screen.

The seventh "Relational Operator" is somewhat different from the other six. It is called a "Substring Operator," and is represented with a dollar sign ($). With it, you can search for a string of characters *or variables* within another string. The correct form of the command reads:

{substring} $ {string}

When instituted, the program searches the {string} on the right for the {substring} on the left.

To see how it works, type in the following:

```
USE People {CR}
LIST FOR "Lover" $ Address {CR}
LIST FOR "4" $ Zip {CR}
```

When you finish, the screen should look like this:

```
USE People ok
LIST FOR "Lover" $ Address
00003   Zachry, Mike   746 Lover's Lane
    Denver      CO   85678   $957.00   ok
LIST FOR "4" $ Zip ok
00004   Dow, Tony       1987-B E. 19th.
    New York    NY   01004   $  4.59
00005   Clark, Bill     657 S. Main
    Portland    ME   00648   $ 34.95   ok
```

The program found the substring "Lover" in the "Address" field of record #00003, and the substring "4" in the "Zip" field of records #00004 and #00005 and returned a logical "True" for those records.

Spend a few moments experimenting with "Relational Operators" as they relate to our sample "People" database.

Logical Operators

"Logical Operators" take a single true-false value, or a combination of true-false values, and returns *one* true or false result.

There are three "Logical Operators" (listed in order of precedence within an expression):

```
.NOT.
.AND.
.OR.
```

As with "Mathematical Operators," parentheses ( ) are used for grouping.

Operators, cont.

"Logical Operators," when used with **H & D Base**, *must* have a blank on either side of them.

These are valid expressions which include "Logical Operators":

```
LIST FOR STATE = "NY" .AND. ZIP = "01004"
```

When entered, the program searchs for any record with "NY" in the STATE field. When it finds one, it looks at the ZIP field for "01004". If it appears, it determines that that particular record delivers a logical "True" to the statement. For any record which does not match *both* of the qualifiers, the program delivers a logical "False" to the statement.

```
LIST FOR ZIP > 50000 .OR. AMOUNT < 10
```

In response to this command, the program searchs for any record with a number in the "ZIP" field which is greater than "50000" *or* any record with a number in the "AMOUNT" field which is less than "10". Any record which satisfies *either* of these statements is deemed to be logically "True". Any record which satisfies *neither* of the statements is deemed to logically "False."

Let's apply these two statements to our "People" database at this time. Enter the following from the COMMAND LEVEL of the program:

```
USE People {CR}
LIST FOR State = "NY" .AND. Zip = "01004" {CR}
LIST FOR Zip > 50000 .OR. Amount < 10 {CR}
```

When you finish, the screen will look like this:

```
USE People ok
LIST FOR State = "NY" .AND. Zip = "85678"
00004   Dow, Tony        1987-B E. 19th.
   New York    NY   01004   $ 4.59     ok
LIST FOR ZIP > 50000 .OR. Amount < 10
00001   Smith, Paul      5487 Oak St.
   San Diego   CA   92376   $267.92
00003   Zachry, Mike     746 Lover's Lane
   Denver      CO   85678   $957.00
00004   Dow, Tony        1987-B E. 19th.
   New York    NY   01004   $ 4.59     ok
```

Record #00004 was the only record in the file for which a logical "True" was generated when it was compared with both sections of the first LIST statement. To the second statement, three records, #00001, #00003, and #00004 were deemed logically "True" because they satisfied the terms of *either* of the LIST statements applied to them.

Take a few moments to experiment with applying "Logical Operators" to the data in our "People" database.

In a manual of this type it is very difficult to fully cover the nature of the "Logical Operators" .AND., .OR., and .NOT. You may want to refer to a computer text with a section on "Logical Operators" in it for further revelations.

String Operators

There are two operators you can use to manipulate "Strings." The are:

1.) + :String concatenation (exact)
2.) — :String concatenation (moves blanks)

In layman's terms the word "concatenation" means taking something and sticking it on the end of something else. As used in **H& D Base**, it means to take one string and put it on the end of another.

Operators, cont.

The correct forms for "String Operation" are:

```
{string} + {string}
```

for "exact" concatenation, and:

```
{string} - {string}
```

for a concatenation which moves trailing blanks of the first string to the end of the combined strings.

"Trailing blanks" are the spaces of a field which are not filled with entered data. For example, the "Name" field of our sample database "People" was designed to hold up to 30 characters. The name "Smith, Paul" in record #00001 of the file fills just 11 characters of the field. There are, therefore, 19 "trailing blanks" in the "Name" field of that record.

Any "trailing blanks" of the first field in this command will appear *between* the strings if you use the " + " form of the command. They will appear *after* the new combined string if you use the "-" command.

To demonstrate, enter the following from the COMMAND LEVEL of **H & D Base**:

```
? "MISSIS      " + "SIPPI" {CR}
? "MISSIS      " - "SIPPI" {CR}
USE People {CR}
? City + State {CR}
? City - State {CR}
```

The screen should look like this when you finish:

```
? "MISSIS        " + "SIPPI"
MISSIS        SIPPI ok
? "MISSIS        " - "SIPPI"
MISSISSIPPI ok
USE People ok
? City + State
San Diego CA ok
? City - State
San Diego CA ok
```

There is a way to *eliminate* trailing blanks altogether. The function which does so is called TRIM, and it is detailed in the next section.

Functions

Some of the functions you will eventually want to perform on your databases are very difficult (or impossible) to perform using the regular arithmetic, logical, and string operators we've discussed in this section. The programmers of **H & D Base** have identified many of these operations and included in the language a number of special FUNCTIONS that you may use.

At this time, we will give you just an overview of the FUNCTIONS available to you. We will not included step-by-step examples for each command. Consult the *Reference* section for further information.

FUNCTIONS can be run from either the COMMAND LEVEL or the PROGRAMMING LEVEL of **H & D Base**. To use one of the FUNCTIONS described below from the COMMAND LEVEL, enter a question mark (?) followed by a space and the FUNCTION. The parenthesis ( ) in the formulas *must* be used.

Blank String Function

SPACE(*nnn*)

This function returns a character string of *nnn* spaces. Example:

```
STORE SPACE(10) TO Mstr
```

The memory variable "Mstr" will now be a 10 character string of blanks.

Date Function

DATE( )

This FUNCTION will generate a character string that contains the date stored in the Atari computer (as established in the DESKTOP) in the format XX/XX/XX. The characters should be entered exactly as shown (without anything between the parentheses).

Decimal Place Function	DEC({numeric expression},{decimal places}) This function sets the decimal position of the {numeric expression} to the number specified in {decimal places}.
Deleted Record Function	* This FUNCTION delivers a logical "True" (.T.) if the current record has been marked for deletion, and a logical "False" (.F.) if has not.
End-of-File Function	EOF This FUNCTION is used to determine if the end of the file has been reached. It delivers a logical "True" (.T.) if it has, and a logical "False" (.F.) if it has not.
File Function	FILE({"filename"/variable/expression}) This FUNCTION will tell you if a certain file exists on the disk. It will generate a logical "True" (.T.) if it is, and a logical "False" (.N.) if it isn't.

Integer Function	INT({numeric expression})
	
	This FUNCTION takes a number with decimals and *eliminates* everything to the right side of the decimal point.
	
	To "round off" a number with a decimal to the nearest whole number, use this form of the INT FUNCTION:
	
	INT(value + .05)

Functions, cont.

Integer to String Function

STR({expression/variable/number}, {length},{decimals})

This FUNCTION converts a number (or contents of a numeric variable) into a string with a specified length and a specified number of digits to the right of the decimal point. The length you specify *must* be large enough to hold at least all the digits plus the decimal point.

Number to Character Function

CHR({number})

This FUNCTION yields the ASCII *character equivalent* of a specified number.

Rank Function

RANK ({string})

This FUNCTION is used to return the ASCII value of the *first* character of a string.

Record Function

#

This FUNCTION delivers the record number of the current file.

String Length Function

LEN({variable/string})

This FUNCTION tells you how many characters there are in the string you name. Note: If a character field variable name is used, it will tell you how many characters are in the *entire* field.

String to Integer Function

VAL({char string})

This FUNCTION converts a character string (or substring) into a number of equal quantity. The string can be made up of digits, a sign, and up to one decimal point.

Substring Function

$({expression/variable/string},{start},{length})

This FUNCTION allows you to select certain characters from a string or character variable. You must specify the starting position and the length.

Substring Search Function

@({variable 1/string 1},{variable 2/ string 2})

This FUNCTION will tell you the beginning position of one string within another string. If the first string does not appear in the second, a value of "0" will be issued.

Trim Function

TRIM ({string})

This FUNCTION is used to eliminate all trailing blanks in the contents of a string variable. The correct form for doing so is:

```
STORE TRIM {variable} TO {new variable}
```

Type Function

TYPE({expression})

This FUNCTION yields a single character string that contains a "C", "N", "L", or "U" depending on whether the data in the expression is "Character", "Numeric", "Logical", or "Undefined" (respectively).

Uppercase Function

!({variable/string})

Functions, cont.

This FUNCTION changes all the *lower case* alphabetic characters in a string or string variable into *upper case* characters.

Function Summary

Each of the FUNCTIONS described in this section is covered in detail in the *Reference* section of this manual. After you have finished reviewing *all* sections of the *Tutorial*, we suggest that you turn to the *Reference* section and work carefully through the examples listed there.

Summary

The *Intermediate Command Tutorial* you have just completed was written to give you just an overview of the forms that **H & D Base** "Expressions" can take. As we said at the beginning of the section, there is no way that we can possibly cover all aspects of their operation. Suffice it to say that just about *any* task you want done can be accomplished in some manner.

For further information, study the *Reference* section or one of the fine books which have been written as companions to the "dBASE II" program. Above all, *experiment* -- it will be your best teacher.

In the next section of the manual, the *Advanced Command Tutorial* we will continue basically where the *Beginning Command Tutorial* left off. Its purpose will be to increase your vocabulary of important **H & D Base** commands.

Chapter 5
ADVANCED COMMAND TUTORIAL

Introduction

This is the *Advanced Command Tutorial*. In it, we will attempt to broaden the scope of what you can do with **H & D Base** by giving you an overview of a number of important system commands.

The first section will cover the procedures for copying databases and for modifying their structures. It will continue with the commands for rapidly altering data in a database, and for placing information in alphabetic or numeric order. Instructions for finding information in a database, generating reports, and performing various arithmetic functions are covered in the latter part of the section, and it concludes with techniques for interacting directly with the system and properly closing all files.

Take your time! Enter each example and make sure you understand how a particular command is used. For further information on *any* command, consult the *Reference* section of the manual.

Copying a Database

The COPY Command

It is possible to duplicate an entire database (data and all) with the COPY command of **H & D BASE**. The most important application of this command is the "back-up" of your data files. *WE STRONGLY ENCOURAGE YOU TO MAKE REGULAR BACK-UPS OF YOUR DATA FILES TO PROTECT AGAINST ACCIDENTAL DATA LOSS.*

The basic form of the COPY command is:

```
COPY [scope] TO {filename}
```

The process is simple and straightforward. While at the COMMAND LEVEL, follow this procedure:

1.) USE the file you want to COPY *from*

2.) Enter "COPY TO" and the name of the new file you want to COPY *to*

The file will be copied (you will be updated as to its progress), and you will be returned to the COMMAND LEVEL.

Note: When you COPY *to* a filename which already exists, all data in that file is *destroyed*.

Let's copy our sample "People" file to a new file named "Temp". Make sure the disk with our "People" file is in the drive and enter the following:

```
USE People {CR}
COPY TO Temp {CR}
```

Once you have entered the second {CR}, the file will be copied and you will be returned to the COMMAND LEVEL. To see the results of your efforts, enter:

```
USE Temp {CR}
DISPLAY ALL {CR}
```

All five records of our sample "PEOPLE" file will appear on the screen.

The COPY STRUCTURE Command

If you would like to copy just the structure of a file to a new one (not the data), simply enter the following from the COMMAND LEVEL:

```
COPY STRUCTURE TO {filename}
```

To demonstrate, type-in the following:

```
USE People {CR}
COPY STRUCTURE TO Temp1 {CR}
USE Temp1 {CR}
LIST STRUCTURE {CR}
```

Note that you now have a new file named "Temp1", but there are no records in it.

Alternate Forms of the COPY Command

There are other forms of the COPY command in **H & D Base** which are available to you. They include:

```
COPY TO {file name} SDF
COPY TO {file name} DELIMITED
COPY TO {file name} STRUCTURE EXTENDED
```

Instructions for their use are included in the "Reference" section of this manual.

Modifying a Database Structure

The MODIFY STRUCTURE Command

Once you have built a database structure with the CREATE command, there is a chance that you will want to alter it in some manner. This can be done with the MODIFY STRUCTURE command. If no data has been entered, the process is a simple one. It is just a bit more difficult if there is information on file.

MODIFY STRUCTURE: Data Not Resident

We will begin our discussion here with the command sequence for altering the structure of a file into which no data has been entered.

Please Note: This sequence of commands should not be used to change the structure of a database into which information has already been entered as it *destroys all data in the file* when it is executed.

The MODIFY STRUCTURE process is carried out as follows:

1.) From the COMMAND LEVEL, USE the file whose structure you want to alter.

2.) Enter the command "MODIFY STRUCTURE."

3.) To the question "Do you want to continue?", enter "Y" for "YES".

4.) The file's structure will appear on the screen. Alter it to your new specifications with the same cursor control and function codes you learned in the EDIT section of this manual. Note: {CTRL-N} will insert a new field in the structure.

5.) When you are finished press {CTRL-W} to save the changes and return to the COMMAND LEVEL of the program.

Let's alter the structure of the new demonstration file "Temp1" that we created in the section immediately prior to this one. If you recall, there is no data in it at this time.

Enter the following from the COMMAND LEVEL:

```
USE Temp1 {CR}
MODIFY STRUCTURE {CR}
```

A list of the fields and their accompanying information will appear on the screen. Let's change the name of the "Zip" field to "Zipcode". Use the cursor control keys to move the cursor to the "Z" in "ZIP", and enter the word "Zipcode" over it. Now press {CTRL-W} to save the change and return to the COMMAND LEVEL of the program.

Take a look at the new structure of our "Temp1" file. Enter:

```
DISPLAY STRUCTURE {CR}
```

Note that the name "Zipcode" has replaced "Zip" in that field.

MODIFY STRUCTURE: Data Resident

The process of modifying the structure of a database into which data has already been entered involves copying the structure of the file to be altered to a temporary file, changing that structure, and transferring the data of the original file to the new one.

Important Note: You *cannot* change the name of an existing field with the process detailed below. If you do, the data from that field will be lost.

If you recall, our sample database named "People" has five records in it. Let's alter it's structure even though it has information in it. Enter the following from the COMMAND LEVEL:

```
USE People {CR}
COPY STRUCTURE TO People1 {CR}
USE People1 {CR}
MODIFY STRUCTURE {CR}
```

Modifying a Database Structure, cont.

The structure of our new file "People1" should be on the screen. Using the cursor control commands, move to the first *blank* record on the screen (field #7). In the corresponding fields, enter this information:

NAME = Due
TYPE = C
WIDTH = 8
DECIMAL PLACES = (No Entry)

Press {CTRL-W} when you are finished. You have just added a seventh field to our database, an 8 digit character field named "Due".

The APPEND FROM Command

To transfer the information from one database to another, you may use a special form of the APPEND command you learned about in the *Beginning Tutorial*. APPEND FROM takes all of the records from one database and adds them to the bottom of a destination database *for every field with a matching name.*

Before executing the APPEND FROM command, we strongly encourage you to make "Back-ups" of *both* files in case of data transfer error. One further word of caution: Do *not* use the "Escape" key to abort the APPEND FROM process; its use could cause serious problems.

Because all of the field names of our source file ("People") appear in our destination file ("People1"), we can execute the APPEND FROM command without the loss of any data. To do so, type:

```
APPEND FROM People {CR}
```

A copy of all the records in the source file is now transferred to the destination file. The information on the source file ("People") remains intact.

Take a look at our new "People1" file with the DISPLAY ALL command. You will find that it consists of five records, each with seven fields (instead of six as in the file "People").

Note: There are two other commands that facilitate the transfer of data from one file to another. They are the UPDATE and JOIN commands and they covered at the end of this *Advanced Command Tutorial* section.

There is one last (optional) step which can be carried out. In most cases you will want to delete the original file and change the name of the new one to match that of the old. You may do so with the DELETE FILE and RENAME commands.

The DELETE FILE Command

To delete a file, simply enter this command:

```
DELETE FILE {filename}
```

Execution of this command will eliminate the designated file and *there will be no way of retrieving it*.

As applied to our on-going demonstration, enter the following from the COMMAND LEVEL:

```
DELETE FILE People {CR}
```

When the process has been completed, you will be prompted on the screen.

Note: If you attempt to DELETE a file with any other "trailer" than ".DAT", you *must* include the "Trailer" in the DELETE statement.

The RENAME Command

The command used for changing the name of a file is RENAME. In its proper form, it reads:

```
RENAME {oldfile} TO {newfile}
```

Modifying a Database Structure, cont.

When we apply this syntax to our example we come up with a command that reads:

```
RENAME People1 TO People {CR}
```

Enter it at this time. You may check to see that the process was properly executed by DISPLAYING the file named "PEOPLE", and *trying* to DISPLAY a file named "PEOPLE1".

Rapid Alteration of Data

In an earlier section of this manual you learned how to edit one record at a time using the EDIT command. There are times, however, when you will want to change the information in a certain field of a record, group of records, or all records to the *same* value (or a value that has been derived from a calculation involving the information in fields of a record). These things can be accomplished with the REPLACE command.

The REPLACE Command

The REPLACE command, in its simplest form reads:

```
REPLACE [{scope}] {field} WITH {exp}
```

In layman's terms, this phrase is saying "Throw out what is in this field of these records, and replace it with this."

Lets take a moment to examine the various elements of this command.

1.) {scope}

Using the {scope} expression in relation to the REPLACE command, you can specify a certain *portion* of the database to be replaced. It can take three forms:

ALL
(Ex: REPLACE ALL)
Replaces a certain field in all records in the file

NEXT *n*
(Ex: REPLACE NEXT 5)
Replaces a certain field in the next 5 records including the current record

RECORD *n*
(Ex: REPLACE RECORD 3)
Replaces only record 3

Rapid Alteration of Data, cont.

2.) {field}

With this portion of the statement you are telling the program which field it is you want to REPLACE.

3.) WITH {exp}

The WITH {exp} statement is used to tell the program what the information is that you want placed in the fields you've chosen above.

Used with the REPLACE command, the WITH {exp} statement could look like any of the following:

REPLACE ALL Zip WITH "99999"
This statement would place the numbers "99999" in the ZIP field of all records.

REPLACE NEXT 3 Address WITH "NONE"
This statement would place the word "NONE" in the ADDRESS field of the next 3 records.

REPLACE RECORD 5 Amount WITH "223.96"
This statement would place the number "223.96" in the AMOUNT field of record #5 *only*.

Try your hand at replacing information at this time. Our sample database, "People", has a field named "Due" which was added to its structure in an earlier lesson. There is no information in the "Due" field of *any* record in the file. Let's put the date "12/31/88" in the "Due" field of all the records.

To do so, enter the following from the COMMAND LEVEL of the program:

```
USE People {CR}
REPLACE ALL Date WITH "12/31/88" {CR}
```

The program will carry out your command with the entry of the last {CR}. You can check its work by entering:

```
LIST Due {CR}
```

There will be nothing on the screen but a column of "12/31/88"'s.

The REPLACE WITH FOR Command

There is another form of the REPLACE command that it will be good for us to review at this point. It is:

```
REPLACE [{scope}] {field} WITH {exp} [FOR {exp}]
```

Note that it is the same as what we've already covered except for the "FOR {exp}" trailer. In layman's terms (once again), it is saying: "Throw out what is in this field of these records, and replace it with this *if this is true*."

Suppose that we wanted to change the due date of all our accounts to "06/01/87" *except* those which are in California. We would enter the following:

```
REPLACE ALL DUE WITH "06/30/87" FOR STATE < > "CA"
```

Hopefully you can begin to see what a powerful statement this becomes with the myriad of command combinations it allows.

Placing a Database In Order

The records you enter into a database are commonly entered in random order. It is possible to put them into either alphabetic or numeric order with two commands available in **H & D Base**. The two commands are SORT and INDEX.

The SORT Command

The SORT command creates *an entirely new file* which has been placed in either ascending or descending order according to the information in one particular (key) field. The correct syntax for the command is:

```
SORT ON {fieldname} TO {filename} [ASCENDING]
                                  [DESCENDING]
```

You may specify any field of a database to SORT on. This is your "Key" field, and it is identified in the above command as {fieldname}. The {filename} portion of the command refers to the name of the new file you want to create.

Ascending or Descending Order

You may SORT a file in either ASCENDING or DESCENDING order by tagging on the appropriate word to the end of the phrase. (Note: If you do not specify that you want the file in either ASCENDING or DESCENDING order, the program will assume that you want it in ASCENDING order).

You may recall that we entered the records of our sample database named "People" in random order. Take a look at it by entering:

```
USE People {CR}
LIST {CR}
```

Placing a Database In Order, cont.

The screen will look like this:

```
USE People ok
LIST
00001   Smith, Paul    5487 Oak St.
    San Diego   CA   92376   $267.92
00002   Jones, Alice   9227 E. Sample #108
    Miami       FL   39857   $ 67.98
00003   Zachry, Mike   746 Lover's Lane
    Denver      CO   85678   $957.00
00004   Dow, Tony      1987-B E. 19th.
    New York    NY   01004   $ 4.59
00005   Clark, Bill    657 S. Main
    Portland    ME   00648   $ 34.95ok
```

Now let's create a file named "Psort" which is a sorted version of the original. We will place the file in alphabetic order using as our "Key" field the one called "Name". Enter the following:

```
SORT ON Name TO Psort {CR}
```

The program will create the new file, SORT the records to it, and return to the COMMAND LEVEL. *The original file will not be altered in any manner.*

To check its work, enter the following:

```
USE Psort {CR}
LIST {CR}
```

Placing a Database In Order, cont.

This screen, reflecting the changes, will appear:

```
USE Psort ok
LIST
00001  Clark, Bill     657 S. Main
   Portland    ME  00648  $ 34.95
00002  Dow, Tony       1987-B E. 19th.
   New York    NY  01004  $  4.59
00003  Jones, Alice    9227 E. Sample #108
   Miami       FL  39857  $ 67.98
00004  Smith, Paul     5487 Oak St.
   San Diego   CA  92376  $267.92
00005  Zachry, Mike    746 Lover's Lane
   Denver      CO  85678  $957.00
```

Note that the records are in sorted order according to the entries in the "Name" field, and that the record numbers have been changed to reflect that order.

Note: You may use the "Escape" key {ESC} to "break out" of the SORT. It will not harm the *original* file in any way.

Sorting on More Than One Field

There are times when you will want to SORT on more than one field. For example, you may want to create a file which has all of your entries in order by "City" within their respective "States".

The correct form for doing so is:

```
SORT ON {fieldname} + {fieldname} TO {filename}
          [ASCENDING] [DESCENDING]
```

To SORT on several key fields, start with the *least important key* and end with the *most important key*.

The correct form for a statement which places a file in order by

"Cities" within "States" is:

```
SORT ON City + State TO Psort {CR}
```

Once you have created a new, sorted version of a file, you might want to DELETE the original and RENAME the new one to its name.

There are a number of drawbacks to the SORT process. Three of the major ones are: 1) The process is very time consuming, 2) Because you are creating an entirely new file, it takes up a great deal of your disk storage space, and 3) If you add any records to a file which has been SORTED, you *must* RE-SORT it if you want the new records placed in their proper positions.

The INDEX Command

You can overcome (to a degree) these problems by INDEXING the file instead of SORTING it. When you enter the command to INDEX a file, the program creates a new file, but places in that file *only* the record numbers of the original file *in sorted order* according to the field you designated. From that point on, when you USE the original file *in relation to the INDEX file*, it will seem as though the file has been SORTED, except the record numbers will not be in order.

The correct form of the INDEX statement is:

```
INDEX ON {key} TO {filename}
```

The "Key" field can be no longer than 100 characters and you cannot INDEX a "Logical" field. You can INDEX on more than one field using the form learned at the end of the SORT section.

Let's take a look at how it works. Our "People" sample file

Placing a Database In Order, cont.

remains in random order (by "Name") at this time. In order to create an INDEX file for it which "keys" in on the "Name" field, enter:

```
USE People {CR}
INDEX ON Name TO Nameind {CR}
```

".NDX" Files

With the entry of this simple command, an INDEX file named "Nameind" is created. All INDEX files carry the trailer ".NDX". Therefore, it you do a directory of your disk our new INDEX file appears as "Nameind.NDX".

Using an INDEX File

The correct form used for telling the program that you want to USE an INDEX in relation to the file from which it was created is:

```
USE {filename} INDEX {indexname} [,{indexname}]...
```

When we place our example in this form, we get:

```
USE People INDEX Nameind {CR}
```

Enter it at this time. Now LIST the program. The screen should look like this:

```
USE People INDEX Nameind ok
LIST
00005  Clark, Bill     657 S. Main
   Portland   ME  00648  $ 34.95
00004  Dow, Tony       1987-B E. 19th.
   New York   NY  01004  $ 4.59
00002  Jones, Alice    9227 E. Sample #108
   Miami      FL  39857  $ 67.98
00001  Smith, Paul     5487 Oak St.
   San Diego  CA  92376  $267.92
00003  Zachry, Mike    746 Lover's Lane
   Denver     CO  85678  $957.00ok
```

Note that everything appears as if the the file had been SORTED, *except* that the record numbers are still in their original entry order.

Some special thoughts on the INDEX command:

1.) If you APPEND, EDIT, REPLACE, or PACK a file while using an INDEX file with it, the program will update the INDEX as you go along to reflect any changes.

    It will continually update *more than one* INDEX at a time if you designate more than one index file with the USE command.

2.) If you execute any of the above commands and you are *not* using any one (or all) of the INDEXES which have been created for the file being manipulated, the INDEXES will have to be recreated.

3.) The fastest way to locate a certain record in a file is to use the FIND command (see below). This command can *only* be used if a file has been INDEXED and that INDEX is being USED in association with it.

4.) You may use the "Escape" key {ESC} to "break out" of the indexing routine. It will not harm the datafile in any way, but the INDEX file will be useless.

5.) An understanding of the "SET INDEX TO" command covered in the "Reference" will be helpful.

Finding Information In a Database

There are two commands you can use to locate a particular record in a file. One can be used at will, while the other must be used only in relation to an INDEX file. The former is the LOCATE command, and the latter is the FIND command.

The LOCATE Command

When you are looking for specific data in a file that is *not* indexed on the "Key" you are interested in, use the LOCATE command. When instituted, it searches a file for the first occurance of a string of characters that you have specified.

The correct form of the LOCATE command is:

```
LOCATE [{scope}] FOR {expression}
```

If you want to search the entire database between the current pointer location and the end of the file, you do not have to specify the {scope}. If you want to search the *entire* file, specify "ALL". In order to LOCATE data in a character field ("C"), your entry should be enclosed in quotes.

The "Zip" field of Record #00004 of our "People" database consists of the numbers "01004" (our {expression}). To LOCATE that record, enter:

```
USE PEOPLE {CR}
LOCATE ALL FOR Zip = 01004 {CR}
```

The program will LOCATE the *first occurance* of "01004" in the "Zip" field and place its accompanying *record number* on the screen. From this point you may either 1) Look at the record with the DISPLAY command or 2) proceed searching for another occurance of "01004" in the file by entering:

```
CONTINUE {CR}
```

If **H & D Base** does not find another match, or if there was no match in the entire file, one of these prompts will appear:

```
END OF LOCATE
END OF FILE ENCOUNTERED
```

You will be returned to the COMMAND LEVEL of the program.

Because our demonstration file contains only five records, the LOCATE function was carried out quite rapidly. You will find, however, that it can be an alarmingly slow process if there are a large number of records in the file. To locate records quickly, use the FIND command.

The FIND Command

The FIND command is very similar to the LOCATE command except that *it can only be used when a database has been indexed and that index file is in USE.* The proper syntax for the command is a simple:

```
FIND {character string}
```

The "character string" can be any number of alpha or numeric characters (it must begin with the first character of the field). You do *not* have to use quotes around it (as you did with the LOCATE command). The "case" of the letters you search for *will* make a difference; e.g., "MIKE" is not a match to "Mike".

Let's locate the number "01004" once again in our sample database using the FIND command. Remember, you must USE the "PEOPLE" database in relation to an index on the NAME field in order for the FIND command to work properly. We will create an index named "Zipind" which "keys" on the "Zip" field of the database.

Finding Information In a Database, cont.

Enter:

```
USE People {CR}
INDEX ON Zip TO Zipind {CR}
USE People INDEX Zipind {CR}
FIND 01004 {CR}
```

The program will search for the first occurance of the numbers "01004" in the "Zip" field of the "People" database and return you to the COMMAND LEVEL if it finds a match. From that point you can 1) View the record with a DISPLAY command, or 2) View any more occurances with a DISPLAY NEXT *n* command.

If no *identical* match is found during a FIND, the prompt "NO FIND" will appear on the screen.

Generating Reports From a Database

Often times you will want to display or print all or part of the information in a database in a form which is much easier to assimilate than that which is produced by a simple LIST or DISPLAY command. This is especially true if the information is to be used by someone who is unfamiliar with how to "read" unformatted data text.

The REPORT Command

With **H & D Base**, you can format the data into a "REPORT" which includes a page heading, the fields you specify, their respective column headings, column totals, etc. The "REPORT" form you create can be stored in a special file and used over and over again to produce the same results (with different information).

We will cover each step of the REPORT process as we create a REPORT with the information in our "PEOPLE" database. To begin, enter the following:

```
USE People {CR}
REPORT {CR}
```

The program will respond with this prompt:

```
ENTER REPORT FORM NAME:
```

".FRM" Files

Enter the name that you want the REPORT "Form" to be stored under. It will become a file on your disk with the trailer ".FRM".

Let's call our REPORT "PReport". Enter it and press {CR}. When you do, these prompts will appear:

Generating Reports From a Database, cont.

```
DEFAULT OPTIONS: LEFT MARGIN (M) = 8, LINES/PAGE (L)
= 57, PAGE WIDTH (W) = 80
ENTER OPTIONS:
```

You are being asked if you would like to change the left margin, lines/page, and page width default settings. The parameters the program defaults to are:

1.) LEFT MARGIN: 8
2.) LINES/PAGE: 57 Lines
3.) PAGE WIDTH: 80 Characters

Your REPORT will be formatted to these defaults if you simply press {CR} at this point. If you would like to change them, they must be entered in this form *before* you press {CR}:

M = {left margin}, L = {lines/page}, W = {page width} {CR}

For example, to change the left margin to "10", the lines/page to "50", and the page width to "70", you would enter:

M = 10, L = 50, W = 70 {CR}

Note: The width is used for centering purposes only.

Do not change the default parameters -- just press {CR}. Once you have done so, a prompt asking whether or not you want a "Page Heading" will come on the screen. A "Page Heading" is a word or phrase which will appear (centered) at the top *of every page* of the REPORT. It's maximum allowable length is 80 characters.

Enter "Y" for "YES". You will then be asked for the heading. Enter "PEOPLE REPORT" and press {CR}.

A prompt asking if you would like your REPORT to be double spaced will now appear. You may either enter "Y" for "YES" or "N" for "NO". For demonstration purposes, enter "N" {CR}.

The next prompt which appears concern "Totals." If you indicate that you do want "Totals", the program will add the contents of a selected numeric field (or fields), and place that total at the bottom of the field's column.

If you answer "YES" to the "Totals" question, a "Subtotals?" question will appear. Answering "YES" to the "Subtotals" question will produce a REPORT which lists subtotals for sets of like records within a database. You will be asked a number of qualifying questions if you enter "Y" to this command. (For more information consult the *Reference* section.)

For demonstration purposes, enter "Y" to the "Totals?" question and "N" to the "Subtotals?" question.

With **H & D Base**, you can specify which fields of the database you would like to include in your REPORT, how much of each field you want to show, and what you want the title at the top of its column to read.

The screen which facilitates these questions looks like this:

```
COL              WIDTH,CONTENTS
001
```

Enter the number of characters of the first field you want to display (any number of characters up to 254). If you enter less than the total possible in the field, it will wrap around to the next line.

Whatever number you enter should be followed with a comma and the name of the field you want listed. It can be any field in the database, regardless of the type. Your entry is recorded with a {CR}.

Before proceeding, you will be asked for the heading you would like centered at the top of the column of information you just set up. Any heading which is less than 60 characters is acceptable (in some instances, part of a heading will wrap around to the next line. A {CR} records your response. If you do not want any heading above the column, simply press {CR} without entering any characters.

Generating Reports From a Database, cont.

Once you have entered the width of field #1, its name, and the column heading, you will be asked if you want to TOTAL the field (if it is a "numeric" field *and* you previously answered "YES" to the "Totals" question).

The same set of questions will be asked for fields #2, #3, and so on. You may put up to 24 fields in a REPORT, but you will find the results distressing if the total number of characters your REPORT requires per line exceeds that which your particular printer is capable of reproducing (usually 80 or 132). Experimentation required...

When you have entered the information for all the columns of your REPORT, press {CR} instead of typing in another column's width. (Note: If you intend to produce a "hard copy" (printed version) of the REPORT, make sure your printer is ready for action). Your REPORT will be generated at this time.

And now for a demonstration. The program should be at a point where it is ready for you to enter the fields of the REPORT. Enter the following information:

Field:	Width:	Name:	Column Heading:
001	30	NAME	NAME
002	20	CITY	CITY
00	37	AMOUNT	AMOUNT

At the beginning of field #004, just press {CR}. This REPORT will appear:

```
PAGE NO 00001
[DATE]

                          PEOPLE REPORT
      NAME                    CITY                    AMOUNT

      Smith, Paul             San Diego               267.92
      Jones, Alice            Miami                    67.98
      Zachry, Mike            Denver                  957.00
      Dow, Tony               New York                  4.59
      Clark, Bill             Portland                 34.95
      **TOTAL**                                      1332.44
```

Notes and Observations:

1.) This file would be in sorted order had we USED it in association with an index "keyed" on the "Name" field before executing the REPORT command.
2.) It is possible to dress this form up to a certain degree. For example, there is a command for justifying titles. For information on how to do so, consult the *Reference* section.

Once you have set-up a REPORT in the manner described above, that REPORT is saved in a file with a ".FRM" trailer eliminating the need to go through this whole process in the future.

The REPORT FORM Command

For instance, our demonstration file is now on the disk under the name "PReport.FRM". You can generate a REPORT from it or any file with the ".FRM" trailer by entering this command:

```
REPORT FORM {formname}
```

As applied to our demonstration REPORT file, the command, when entered, reads:

```
REPORT FORM PReport {CR}
```

For details on its application, consult the *Reference* section of this manual.

Counting Database Records and Totaling Their Contents

It is a simple matter to see how many records there are in any given database. Just enter the command "DISPLAY STRUCTURE" and look for the prompt which reads "NUMBER OF RECORDS". It's listed right there.

The COUNT Command

Suppose, however, that you need to COUNT the number of records in a database which meet a specified condition (or conditions). This is also possible with **H & D Base**.

The full form for the COUNT command is:

```
COUNT [{scope}] [FOR {conditions}] [TO {memory variable}]
```

This command can be used with none, some, or all of the modifiers deliniated by brackets. If you enter just the word COUNT {CR}, it counts all the records in the database. The {scope} can be limited to one or a specified number of records, and the {condition} can be any complex expression (as discussed in the *Intermediate Command Tutorial*). The result of the COUNT is displayed on the screen, and can also be stored in a {memory variable}.

There are five records in our sample database named "People". Three of the five have numbers in the AMOUNT field which are greater than "100". Let's see if we can get the program to COUNT these records.

Enter the following:

```
USE People {CR}
COUNT FOR Amount > 100 {CR}
```

In just a few seconds, the result of the COUNT will appear on the screen in this form:

```
COUNT = 00003
```

You may stop the COUNT at any point be pressing the "Escape" key {ESC}.

The SUM Command

The COUNT command can be used only to determine the number of records in a file which satisfy a certain conditon. There is another command which adds together the numbers in a particular *numeric* field (or fields) in *all* (or specified) records. The command is SUM.

The correct syntax for the entire SUM command is:

```
SUM [{scope}] {exp} [,{exp}] [FOR {condition}]
                    [TO {memvar}]
```

We have but one numeric field ("Amount") in our sample database named "People". To get a total of the numbers in *all* the "Amount" fields of our file, enter:

```
USE People {CR}
SUM Amount {CR}
```

The computer will work for a moment, then display the total on the screen (in this case, "1332.44").

You may total up to five *numeric* fields. If more than one field is being totaled, separate each with a comma. The records totaled can be limited by using the {scope} and/or conditional expressions after the "FOR".

The SUM function can be brought to a halt at any point by pressing the "Escape" key {ESC}.

Counting Database Records and Totaling Their Contents, cont.

The TOTAL Command

There is one more command provided by **H & D Base** which can be used for adding up numbers in a database. The command is TOTAL. Although you will not be given step-by-step instructions on how to use it at this point, it is important for you to know that it exists.

The TOTAL command is used primarily for eliminating detail and providing summaries of information. It works very much like the "Sub-Total" feature of the REPORT command, except that the results are placed in a database rather than being printed out.

For details on the TOTAL command, see the *Reference* section.

Working With Multiple Databases

The work you have done with **H & D Base** to this point has been performed in relation to only one data file (.DAT) at a time. You have learned to specify precisely *which* data file with the "USE {file}" command.

To work with a different database, you simply entered another "USE {file}" command. Upon doing so, the first file was closed, and the second was opened. The record "pointer" was placed at the first record in the newly-opened file.

There will be times when you will want to USE a second file *without losing your place in the first*. In other words, you will want to have two data files open *at the same time*. This is possible with **H & D Base**.

Primary and Secondary Files

The first data file you open with a USE command is called the "PRIMARY" file. The second is appropriately called the "SECONDARY" file. The sequence of commands for USING a "PRIMARY" file and then a "SECONDARY" file is:

```
USE {name of primary file}
SELECT SECONDARY
USE {name of secondary file}
```

The SELECT Command

After establishing a "PRIMARY" and "SECONDARY" file in this manner, you can toggle between the two with this command:

```
SELECT [PRIMARY or SECONDARY]
```

From this point, the commands you issue will be performed in relation to either the "PRIMARY" or "SECONDARY" data file, depending upon which one you choose with the SELECT statement.

Information *can* be transferred from one area to the other using "P." and "S." as prefixes for field names. If you are in the "PRIMARY" area, use the "S." prefixes for field names you need from the SECONDARY area. If you are in the "SECONDARY" area, use the "P." prefix for field names you need from the "PRIMARY" area.

For example:

```
USE People                (The PRIMARY file)
SELECT SECONDARY
USE Product               (The SECONDARY file)
SELECT PRIMARY
  DISPLAY Name            (A field in the PRIMARY file)
  DISPLAY S.Product       (A field in the SECONDARY file)
  DISPLAY Amount          (A field in the PRIMARY file)
```

The JOIN Command

You can combine two databases (the PRIMARY and SECONDARY files) to create a third database. This is accomplished through the use of the JOIN command. In its full form, it reads:

```
JOIN TO {newfile} ON {exp} [FIELD {list}]
```

When executed, this command positions the "pointer" on the first record of the PRIMARY use file and then evaluates, one-by-one, *every* record of the SECONDARY use file to see if it matches the specified "expression." For every match, a new record is added to a completely separate third database ({newfile}).

The optional list of FIELDS that you may add to the basic statement specifies which FIELDS from the PRIMARY and/or SECONDARY files you want transferred to the {newfile}. You may specify any FIELD included in either (or both) USE files. Note: We suggest that you preface the name of *each* FIELD with either a "P." or "S." depending upon whether the FIELD is in the PRIMARY or SECONDARY use file.

If you choose *not* to list the specific fields you want included in the {newfile}, the program will transfer *all* the fields of *both* files (including fields with the same names).

Here is a sample JOIN command sequence:

```
USE People
SELECT SECONDARY
USE Product
JOIN TO Third FOR State= "CA" FIELD P.Name,
   P.Address,P.City,P.State,P.Zip,S.Product,P.Amount
```

Two special notes regarding the JOIN command:

1.) Depending on the size of the two databases involved, this command can take an extremely long time to execute.

2.) If the "expression" you use is too "loose" (e.g., its parameters are too large), there may be a very large number of "matches" that are made. This may cause size of the {newfile} to increase to unmanageable levels.

The UPDATE Command

Data can also be transferred from one database file to another using the UPDATE command. This type of update does *not* use the PRIMARY/ SECONDARY format.

The basic format of the command is:

```
UPDATE FROM {database} ON {Key} [RANDOM]
  [ADD {field list}] [REPLACE {field list}]
  [REPLACE {field} WITH {from field}]
```

Prior to running this command, both database files to be used (the USE database file and the FROM database file) *must* be in order using the *same* {key} (unless the RANDOM statement is used -- see below). The USE database file can be either SORTED or INDEXED, while the FROM database *must* be SORTED.

Working With Multiple Databases, cont.

Assuming that we are working with a USE database file named "People", and a FROM database file named "Product", preparation for running the UPDATE command would be as follows:

```
USE Product
SORT ON Product TO Prodtemp
DELETE FILE Product
RENAME Prodtemp TO Product
USE People
INDEX ON Name TO Nameind
USE People INDEX Nameind
UPDATE FROM Prodtemp TO...
```

Although greatly simplified, this is what occurs when the command is run (with all SORTS or INDEXES completed and in USE): The program compares the contents of the {key} in the USE database file against the contents of the {key} in the FROM database file. If the contents are exactly the same, the remainder of the command is executed on the field (or fields) of the USE database file. If the information in the {keys} do *not* match, the record is skipped and the search for matches continues.

When {key} matches are found, there are three options which can be performed. We will review each briefly.

The "ADD {*field list*}" option will *add* the contents of a specified field (or fields) in the FROM database file to the contents of a field (or fields) *with the same name* in the USE database file. An example:

```
UPDATE FROM Invoice ON Name ADD Total
```

The "REPLACE {*field list*}" option will *replace* the contents of a specified field (or fields) in the USE database file with the contents of a field (or fields) *with the same name* in the FROM database file. An example:

```
UPDATE FROM Invoice ON Name REPLACE Rec:By
```

The "REPLACE {*field*} *WITH* {*from field*}" option will *replace* the contents of a specified field (or fields) in the USE database file with the contents of *any* field (or fields) in the FROM database file.

```
UPDATE FROM Invoice ON Name REPLACE Last:Act
                                                WITH Date
```

The RANDOM Option

If you choose to "tack on" the optional "RANDOM" statement to the end of the entire command, the FROM database file does not have to be SORTED. The USE database file, however, *must* be INDEXED (not SORTED) and that INDEX must be in USE. Although it accomplishes the task in a different manner, the results of including the RANDOM statement in the UPDATE command are the same in all respects.

Changing H & D Base Characteristics

H & D Base provides a number of "SET" commands which control how the program interacts with your system. Some SET commands are merely toggle switches which register a default of either "ON" or "OFF". Others allow you to enter specific values. *All* SET commands can be quickly and easily changed from either the COMMAND LEVEL of the program or from within a COMMAND FILE.

Each of the SET commands which serves as a simple "ON/OFF" toggle, along with its default value and a brief description, is listed below.

SET BELL ON/OFF

(Default = ON)

ON:	Enables the bell which rings when a field has been filled during various functions or when data of the wrong type is entered
OFF:	Disables the bell
NOTE:	The volume of the bell can be altered from the Control Panel of the Atari DESKTOP

SET CARRY ON/OFF

(Default = OFF)

ON:	Carries data from the record just entered forward to the new record when in APPEND
OFF:	Does not carry data forward -- shows just blank record
NOTE:	Used for entering a large number of records which contain the same information in many of the fields

Changing H & D Base Characteristics, cont.

SET CONFIRM ON/OFF

(Default = OFF)

ON: When a field has been filled in APPEND, EDIT, or READ, waits for a {CR} to be issued from the keyboard before continuing

OFF: Continues to next field automatically when current field is full

SET CONSOLE ON/OFF

(Default = ON)

ON: All output is sent (echoed) to the screen

OFF: No output is sent to the screen

SET COLON ON/OFF

(Default = ON)

ON: Uses colons on the screen to show the length of a field during AT...GET, APPEND, and EDIT functions

OFF: No colons on the screen

SET DELETED ON/OFF

(Default = OFF)

ON: Records marked for deletion with an asterisk cannot be located with the FIND commands nor processed by any command that allows the NEXT phrase (such as LIST, LOCATE, COUNT, etc.)

OFF: Records marked for deletion *can* be located and displayed (but not COPIED or APPENDED)

SET ECHO ON/OFF

(Default = OFF)

ON: All commands which come from COMMAND FILE are also being sent to the screen

OFF: Commands from COMMAND FILE are *not* being sent to the screen

Changing H & D Base Characteristics, cont.

SET EJECT ON/OFF	(Default = OFF)	
	ON:	The REPORT command will eject a new page before beginning a new report
	OFF:	A new page will not be ejected
SET ESCAPE ON/OFF	(Default = ON)	
	ON:	Allows a user to use the "Escape Key" {ESC} to break out of the execution of a COMMAND FILE
	OFF:	Disables the "Escape Key"
SET EXACT ON/OFF	(Default = OFF)	
	ON:	Requires that character strings match *exactly* (excluding trialing blanks) in expressions and the FIND command
	OFF:	Allows matches to be made on the basis of the length of the second string
	NOTE:	When "OFF", "ABC" is a match to "ABCDEFG"
SET FORTH ON/OFF	(Default = OFF)	
	ON:	FORTH Program Language commands available
	OFF:	FORTH Program Language commands not available
SET LINKAGE ON/OFF	(Default = OFF)	
	ON:	Moves the record pointers in PRIMARY and SECON DARY files *simultaneously* (in increments) in association with commands that allow {scope} (LIST, REPORT, SUM, etc.)
	OFF:	Makes PRIMARY and SECONDARY record pointers work independent of one another

SET PRINT ON/OFF

(Default = OFF)

ON:	All output is sent (echoed) to the printer
OFF:	Output is not sent to the printer

SET RAW ON/OFF

(Default = OFF)

ON:	DISPLAYS and LISTS records without spaces between fields
OFF:	DISPLAYS and LISTS records with an extra space between fields

SET TALK ON/OFF

(Default = ON)

ON:	Displays the results of commands on the screen
OFF:	Does *not* display the results of commands on the screen

The following SET commands require that you enter a certain value (as opposed to simply toggling between "ON" and "OFF"):

SET ALTERNATE TO {filename}

The SET ALTERNATE TO command is part of a two-step process for writing *everything* that is normally written on the screen to a disk file as well. The file will be stored with a ".TXT" trailer, indicating that it is a "Text File."

To establish a "Text File," enter the following:

```
SET ALTERNATE TO {filename}
```

Changing H & D Base Characteristics, cont.

When you want *all* of the information appearing on the screen to also be sent to the specified Text File, enter:

```
SET ALTERNATE ON
```

To turn the flow of information "off," enter:

```
SET ALTERNATE OFF
```

The information in a text file can be edited, printed, etc. with the aid of a word processor or text editor.

SET COLOR TO {number of character color}

(Default = 0)

Text can be displayed in four different colors using the Atari ST series of computers. The colors, assuming that you have not altered them from the Control Panel of the DESKTOP, are white (the default color), black, red, and and green. They will appear as different shades of grey on a black and white monitor.

To change the color of the characters being typed or displayed on the screen, enter "SET COLOR TO" followed by one of the following numbers:

0 = White
1 = Red
2 = Green
3 = Black

All text entered or displayed from that point on will appear in the color you specified. Note: You may not be able to see black letters at all.

SET DATE TO {mm/dd/yy}

(Default = System Date)

The DATE function of **H & D Base** displays the date stored *in the Atari system*. The command for doing so is:

```
SET DATE TO {mm/dd/yy}
```

The date must be entered in the form shown (mm = month, dd = date, yy = year).

When you enter a date with this command, you are resetting the date stored in the Atari.

SET DECIMAL TO {*n*}

In REPORT, the default number of decimal places used in a numeric field is 2. You may change that number with this SET command.

SET DEFAULT TO {drive}

{Default = Boot Drive}

Whenever you enter the name of a file to be used in relation to a certain command, **H & D Base**, assumes that the file resides on the data disk in the drive that the program was loaded from. You may indicate to the program that a file resides on a disk in an alternate drive by prefacing the file name with the letter which represents that drive followed by a colon.

For example:

```
USE People        (File located on default drive)
USE B:People      (File located on "B" drive)
```

The "SET DEFAULT" command allows you to choose an alternate disk drive that the program will access when searching for files which appear *without* a designated drive letter preceeding them.

A sample "SET DEFAULT" command:

```
SET DEFAULT TO C:
```

The trailing colon is optional.

Changing H & D Base Characteristics, cont.

SET FORMAT TO [**SCREEN**] [**PRINT**] [{**format file**}]	(Default = SCREEN) SCREEN: Sends output of @ commands to the screen PRINT: Sends output of @ commands to the printer {format file}: Uses format previously created for APPEND, EDIT, INSERT, and CREATE commands

SET HEADING TO
{**string**}
	This SET command adds a second page heading to a REPORT. It can be up to 60 characters long.

SET INDEX TO
{**index file**}
[,{**index file**},...
{**index file**}]
	This command opens (or creates and opens) up to seven index files for use in relation to the database file currently resident. It can be used to keep a number of INDEXES updated during functions like APPEND, EDIT, etc.
	
	The first index file named is considered to be the "Master Index." The database file will be in indexed order according to it, and it is the one which will be used in all FINDS.
	
	If an index file is already in use when this command is instituted, that index file will be closed before the new one is opened.
	
	If you enter just "SET INDEX TO {CR}", all indexes will be released.

SET MARGIN TO
{***nnn***}
	This command can be used to change the left margin on the printer during the printing of a REPORT. The number entered can be no larger than "254".

Resetting the Program

When you first enter the **H & D Base** program, all files are closed and no memory variables are active. As you use the program, however, miscellaneous files *are* opened, and memory variables *are* established.

There will be times when you will want to make sure that all the information in the database files you have been using is properly stored. One of those times is when you leave the program; another is when you have data that you are simply afraid of losing; the third is when you want to start over without leaving the program.

There are two ways that you can close all files and release all memory variables. The first is to simply leave the system with the QUIT command (discussed at the start of the *Beginning Command Tutorial*). The second is by entering this command:

```
CLEAR
```

This command closes all database files and releases all memory variables. Use it whenever you want to proceed with a "clean slate".

An alternate form of the CLEAR command is "CLEAR GETS". Its use is discussed in the *Reference* section of this manual.

Summary

This concludes the *Advanced Command Tutorial.* In it you have been shown how to use some of the more advanced statements of the **H & D Base** programming language.

In many cases, you were shown only the most basic applications of the words we covered. To make use of the full of **H & D Base** in regard to these (and other) commands, you *must* study the *Reference* section of this manual carefully.

In the next section, we will show you how to set up **H & D Base** COMMAND FILES (programs), so that you can automate your information process.

Chapter 6
PROGRAMMING TUTORIAL

Introduction

To this point, you have learned how to use a number of powerful commands which allow you to manipulate the information in your databases in a limitless number of ways. In using the commands, you were allowed to enter, and the computer acted upon, just one instruction at a time. The process was somewhat awkward, and time consuming to be sure.

The real power of **H & D Base** comes into play when you form the commands you have learned into a program (or COMMAND FILE) which, when run, accomplishes the same task or purpose over and over again.

This section will teach you how to put the commands you have learned in the proper order, how to make choices regarding them, how to get certain instructions to repeat themselves, and how to access sub-files of commands.

Establishing a Command File

The Nature of a Command File

H & D Base provides users with a method of saving a set of frequently used command sequences in order to alleviate the necessity of entering but one command at a time. A set of command sequences is called a COMMAND FILE and is saved on the disk with a ".CMD" trailer.

When executed with a DO command, the program starts with the first statement, executes it, then continues in a likewise manner until the end of the file is encountered.

The MODIFY COMMAND Command

COMMAND FILES can be created and modified with most text editors and/or word processors which produce standard "text files," and one such "sub-program" has been included in **H & D Base** for that express purpose.

You can access the elementary text editor of **H & D Base** by typing in the following command:

```
MODIFY COMMAND {file}
```

Let's start forming a COMMAND FILE named "Comfile1" at this time. To do so, enter the following:

```
MODIFY COMMAND Comfile1 {CR}
```

When you do, a screen with a dashed line across the top will appear. This signifies that you have entered **H & D Base's** "Command File Editor."

The Command File Editor

Upon entering the MODIFY COMMAND statement from the COMMAND LEVEL of the program, you automatically enter what is referred to as the "Command File Editor." It is actually a program within a program and has been designed expressly for the purpose of entering and editing COMMAND FILES. You may liken it to a *very* simplistic word processor.

To enter a program with the "Command File Editor," type in a line of code and press {CR}. If you attempt to enter more characters on a line than will fit without pressing {CR}, the program will issue the carriage return for you and place the cursor at the beginning of the next available line. (Note: Remember that a "tilde" sign (~) *must* appear at the end of each line for any single command which must continue on the succeeding line.)

Cursor Movement You may use the following cursor movement commands while in the "Command File Editor":

1.) {UP ARROW} or {CTRL-E}
Moves the cursor one line up

2.) {DOWN ARROW} or {CTRL-X}
Moves the cursor one line down

3.) {LEFT ARROW} or {CTRL-S}
Moves the cursor one character to the left

4.) {RIGHT ARROW} or {CTRL-D}
Moves the cursor one character to the right

5.) {HOME} or {CTRL-O}
Moves the cursor to the first character in the Command File

6.) {CTRL-K}
Moves the cursor to the last character in the Command File

The Command File Editor, cont.

In addition, you may use your "mouse" for positioning the cursor.

Miscellaneous Key and Control Functions

There are a number of functions in the "Command File Editor" which are activated through the use of "Control Key" sequences. They are as follows:

1.) {CR}
Inserts a line

2.) {CTRL-Y}
Deletes a line and moves all lines up

3.) {CTRL-G} or {DELETE} or {RIGHT BUTTON ON MOUSE}
Delete character under cursor

4.) {BACKSPACE} or {LEFT BUTTON ON MOUSE}
Delete character to left of cursor

5.) {UNDELETE}
Undelete last *Backspaced* character

This command will "undelete," beginning at the current cursor position, a character or set of characters which have been deleted with the "Backspace" key *only*. Use it for moving all or part of a command from one line to another. (Limit: 256 characters)

6.) {CTRL-W}
Exit - Save

7.) {CTRL-Q}
Exit - No Save

Note: Some of the control functions you will learn about in other sections of this manual will *not* work in the "Command File Editor." The list above is the *complete* list of applicable commands.

Entering a Command File

To get you started in learning how COMMAND FILES work and how the Command Editor operates, let's enter a COMMAND FILE at this time. While in the Command File Editor, type the following *exactly as it appears*:

```
USE People {CR}
LIST {CR}
COUNT FOR Zip>50000 TO Mzipnum {CR}
? {CR}
DISPLAY "THERE ARE " Mzipnum " RECORDS IN THE FILE
   WITH ZIPCODES LARGER THAN 50000" {CR}
? {CR}
DISPLAY "WE DID IT!" {CR}
RETURN {CR}
```

When you have finished, return to the COMMAND LEVEL of the program by pressing {CTRL-W}. In pressing {CTRL-W}, the contents of the Command File Editor is stored on the disk under the name you specified when you entered that mode (along with a ".CMD." trailer). You should now have a file on your disk named "Comfile1.CMD".

Running a Command File

The DO Command

The use of any COMMAND FILE can be instituted from the COMMAND LEVEL of the program with this statement:

DO {command file name}

In entering the COMMAND FILE name, you do *not* have to include the ".CMD" trailer.

Let's run our COMMAND FILE named "Comfile1.CMD" at this time. To do so, enter the following:

DO Comfile1 {CR}

Each of the commands we registered will be executed in the order they appear, producing the following on the screen:

```
DO COMFILE1
00001   Smith, Paul    5487 Oak St.
    San Diego   CA   92376   $267.92
00002   Jones, Alice   9227 E. Sample #108
    Miami       FL   39857   $ 67.98
00003   Zachry, Mike   746 Lover's Lane
    Denver      CO   85678   $957.00
00004   Dow, Tony      1987-B E. 19th.
    New York    NY   01004   $  4.59
00005   Clark, Bill    657 S. Main
    Portland    ME   00648   $ 34.95
THERE ARE 2 RECORDS IN THE FILE WITH ZIPCODES LARGER THAN
50000

WE DID IT! ok
```

If you made any mistakes in entering the program, the system more than likely returned an "Error Message" on the screen. If that happened, re-enter the Command File Editor, correct the problem, and re-run the program from the COMMAND LEVEL using the "DO" statement.

Entering Notes, Remarks and Text

There are a number **H & D Base** commands which allow you to include comments in the body of your program. Some will be displayed on the screen when the program is executed, and some won't.

The NOTE and * Commands

There are two commands which are used for placing comments in a program which are *not* to be displayed when that program is run. They are:

NOTE and *

When either one appears as the *first* entry on *any* line in a COMMAND FILE, the rest of that line is ignored when the program is run. For example, our sample COMMAND FILE ("Comfile1") might be set-up as follows:

```
NOTE *****This program counts records in the*****
NOTE *****file with zips larger than 50000*****
USE People {CR}
LIST {CR}
COUNT FOR Zip>50000 TO Mzipnum {CR}
? {CR}
DISPLAY "THERE ARE " Mzipnum " RECORDS IN THE
   FILE WITH ZIPCODES LARGER THAN 50000" {CR}
? {CR}
DISPLAY "WE DID IT!" {CR}
* This is the end of the routine
```

Note: The asterisk (*) is often used by programmers to temporarily disable a line of commands. Remember, anything which appears after it is ignored when the program is run. (A blank space *must* appear between the asterisk and the first character of the line.)

Entering Notes, Remarks and Text, cont.

Two ways have been provided in **H & D Base** for you to place text (only) in your programs that *will* be displayed on the screen when the program is run. Which one you use depends primarily upon the length of the text you want included.

The REMARK Command

For *short* amounts of text that you want displayed on the screen (or printer), use the REMARK command. When it appears as the first word on a line, anything *which follows it* will be displayed on the screen when the program is run. The string does *not* have to be enclosed in quotes. (Remember that a total command line can be no longer than 255 characters.)

For example, if these two lines were included in a command file:

```
REMARK Are you getting tired yet?
REMARK "Are you getting tired yet?"
```

they would be displayed as:

```
Are you getting tired yet?
"Are you getting tired yet?"
```

when the program is run.

Note: REMARK is very much like this statement:

```
? {"string"}
```

The major difference is that in the statement above, you *must* use quotation marks around the text to be displayed. With the REMARK command, they are not needed.

The TEXT Command

If you have a *large* amount of text you would like to have displayed when your program is run, use the TEXT command. When entered, the word "TEXT" must appear on a line by itself. The characters you want displayed on the screen or printer should begin on the next line of the program. They may con-

Entering Notes, Remarks and Text, cont.

tinue through an unlimited number of lines. To conclude the sequence, enter the command ENDTEXT on the first line below the last characters of your text.

For example:

```
TEXT
Four score seven years ago, our Fathers brought
forth upon this continent a new nation.
ENDTEXT
```

will appear as the following when the program is run:

```
Four score seven years ago, our Fathers brought
forth upon this continent a new nation.
```

Making Choices and Decisions

The IF...ELSE Command

There will be many times within a program that you will want certain functions executed *only* if specific conditions are met. In other instances, you might want to establish two (or more) different program directions depending upon how the data is interpreted.

It's like saying, "If it is sunny today then I am going to the lake." Or, in the second sense, "If it is sunny today I am going to the lake, if not, I am going to stay home and read a book."

The command which facilitates this decision-making function is IF...ELSE. In it's full form, it reads:

```
IF {exp}
   {commands}
[ELSE
   {commands}]
ENDIF
```

The manner in which you apply the formula will depend on whether you are making a simple decision or one to which there will be two or more alternative responses.

Simple Decisions

If, at some point in your COMMAND FILE, you would like the program to make a simple decision to which only one course of action may be taken, you need not use the optional "ELSE" statement *at all*. The command would look like this:

```
IF {exp}
   COMMAND #1
   COMMAND #2
   COMMAND #3...
ENDIF
```

When the "IF" statement is encountered by the **H & D Base** program, it will determine if the "expression" {exp} is logically "True" or "False." If it is "True," the program will execute any command which falls between it and the "ENDIF" statement. If it is "False" it will *skip* any command falling between the "IF" and "ENDIF" statements and continue from that point.

Here is an example using our sample "PEOPLE" file:

```
USE People
IF State = "CA"
   DISPLAY Name
ENDIF
```

When run with a "DO" command from the COMMAND LEVEL of **H & D Base**, this program will display the contents of the "Name" field in the first record of the file *if* the letters "CA" appear in the "State" field.

You may place as many commands as you please between "IF" and "ENDIF" statements, and they do *not* have to be indented. You may find it helpful to do so, however, as an aid for determining the beginning and ending points of the IF...ENDIF statement.

Two Choice Decisions

If there are two different courses of action that depend upon a certain condition or conditions being "True" or "False," you must insert an "ELSE" statement somewhere between "IF" and "ENDIF".

Consider the following example:

```
USE People
   IF State = "CA"
   DISPLAY Name
ELSE
   DISPLAY Amount
ENDIF
```

When run with a "DO" command from the COMMAND LEVEL of **H & D Base**, this program will display the contents of the

Making Choices and Decisions, cont.

"Name" field in the first record of the file *if* the letters "CA" appear in the "State" field. If the letters "CA" do *not* appear in the "Name" field, the program will display what is in the "Amount" field.

Multiple Choice Decisions

At times, you will want your program to make a choice from a list of alternatives (as in a "menu"). This is made possible in **H & D Base** through the use of "nested" "IF...ELSE...ENDIF" statements.

A "nested" statement is one whose beginning *and* end fall *completely* within the confines (beginning and end) of another like statement. In this statement, there are two "nested" "IF...ELSE" statements:

```
IF {exp}
   COMMAND #1
   COMMAND #2 ...
ELSE
   IF {exp}
      COMMAND #1
      COMMAND #2 ...
   ELSE
      IF {exp}
         COMMAND #1
         COMMAND #2 ...
      ELSE
         COMMAND #1
         COMMAND #2 ...
      ENDIF
   ENDIF
ENDIF
```

This particular form could be used in a menu to which there were four options (A, B, C, & D). A different course of action by the program for each choice.

Note: You must have an "ENDIF" for each "IF" or the program will not work properly.

Repeating a Process

The DO...WHILE Command

There will undoubtedly be times when you'll want a program (or part of a program) to repeat itself without intervention on your part. This is easily accomplished in **H & D Base** with this command:

```
DO WHILE {exp}
    [COMMAND #1]
    [COMMAND #2]
    [COMMAND #3]
ENDDO
```

When the DO...WHILE command is encountered by **H & D Base**, it evaluates data against the expression and delivers a logical "True" or a logical "False." If the result is a "True" value, any command between the DO WHILE and ENDDO are performed. When the ENDDO is reached, flow of the program goes back to the DO WHILE command and data is evaluated against the "expression" once again. The process continues until a "False" value is returned by the DO WHILE statement at which point the program skips to the ENDDO and proceeds from there.

Note: If a value of "False" is never delivered by the "DO WHILE" statement, the program will loop forever.

Here is a sample DO WHILE statement:

```
USE People
DO WHILE State = "CA"
    DISPLAY
    SKIP
ENDDO
```

This program will list the records of our sample "People" file with the letters "CA" in the "State" field until a non-"CA" field is reached.

Repeating a Process, cont.

The LOOP Command

If you would like to return to the DO...WHILE statement before reaching the ENDDO, use the LOOP command.

For example:

```
USE People
DO WHILE State = "CA"
    IF Zip< >92376
    LOOP
    ENDIF
DISPLAY
ENDDO
```

This program will list all records of our sample "PEOPLE" file with the letters "CA" in the "State" field and any numbers besides "92376" in "Zip" field.

Note: It is generally an unwise practice to use the LOOP command as it is difficult to track program "flow" when it is resident.

Using the DO WHILE Command as a Counter

If you want a process to be repeated just a certain number of times, create a variable to serve as a counter and add "1" to it each time through. For example:

```
STORE 1 TO Mcounter
DO WHILE Mcounter>20
    [COMMAND #1]
    [COMMAND #2]
    [COMMAND #3]
    STORE MCounter + 1 TO Mcounter
ENDDO
```

Nested Command Files

Nested Do Commands

H & D Base allows you to open and run a COMMAND FILE (or files) from within another COMMAND FILE. To carry it one step further, you may also run a COMMAND FILE within a COMMAND FILE, within a COMMAND FILE, within a COMMAND FILE, and so forth (up to 16 levels deep). This is accomplished by "nesting" the DO statement for one file within the statements of another.

For example, you can design a COMMAND FILE which is nothing more than just a menu which initializes the running of a number of separate COMMAND FILES (one for each possible choice on the menu). The part of the program we are interested in might look like this:

```
IF Mchoice = "A"
   DO Genledger
ELSE
   IF Mchoice = "B"
      DO Receive
   ELSE
      IF Mchoice = "C"
         DO Payable
      ELSE
         IF Mchoice = "D"
            DO Payroll
         ENDIF
      ENDIF
   ENDIF
ENDIF
```

There are two ways that the execution of a COMMAND FILE is halted and control is returned to the COMMAND FILE which "called" it:

1.) The end of the file is reached

2.) A "RETURN" statement is encountered

Nested Command Files, cont.

The RETURN Command

The RETURN command "closes" a COMMAND FILE and returns control to the COMMAND FILE that called it. If it was *not* called by another COMMAND FILE, control returns to the COMMAND LEVEL of the program.

The RETURN command can appear *anywhere* in a COMMAND FILE. It is not always necessary for you to include a RETURN statement at the bottom of COMMAND FILES, but it is a good programming practice.

Entering Data During the Run of a Program

In many instances, the programs you write will require a certain degree of interaction with the people running them. You may ask them to enter specific data or register their choice to a number of options (as in the menu sample in the last section).

There are three commands which **H & D Base** provides for you as a programmer to use in such cases. They are:

1.) WAIT [TO]

2.) INPUT TO

3.) ACCEPT TO

We will cover each command briefly at this time.

The WAIT Command

WAIT [TO {memvar}]

The WAIT command brings to a complete halt the operation of a program and places the word "WAITING" on the screen. Operation of the program resumes when *any single key* is depressed.

If the command appears along with the "TO {memvar}" option, the character inputed by the user is stored to the {memvar} specified. If the character is one which is non-printable (such as {CR}), a blank is entered into the variable.

Example:

```
WAIT TO Mtemp
```

Entering Data During the Run of a Program, cont.

The INPUT Command

INPUT ["prompt"] TO {memvar}

The INPUT command, like the WAIT command, stops the operation of a program and waits for information to be entered by the user. Instead of a "WAITING" prompt, however, you as the programmer have the option of defining your *own* customized prompt. The prompt will appear on the screen.

An additional difference is that more than just one character (of any data type) may be entered and stored to the variable. The type of variable it is will be determined by the nature of the information entered. *Character strings must be entered in quotes or square brackets.* The INPUT command is used most commonly for the entry of numbers.

Example:

```
INPUT "ENTER THE NUMBER OF NAILS HERE" TO Mnails
```

The ACCEPT Command

ACCEPT ["prompt"] TO {memvar}

The ACCEPT command operates almost exactly like the INPUT command with one notable difference. *Any* type of data may be entered (no quotes are necessary), although all information will be stored as *character* data. Note: Numbers should be entered using the INPUT command.

Example:

```
ACCEPT "ENTER THE ADDRESS HERE" TO Maddress
```

Displaying Text and Data at Alternate Screen Locations

The prompts which are associated with the "?", "INPUT", and "ACCEPT" commands we have covered so far are displayed by the program flush to the left margin of the screen, one line below the last entry. It is possible (and quite simple) with **H & D Base** to move those prompts to alternate screen locations.

Row, Column Coordinates

The screen of your Atari ST computer is divided into 25 rows (horizontally) and 80 columns (vertically). Rows are numbered consecutively from "0" to "24", and Columns are numbered consecutively from "0" to "79". Each intersection of a Row and Column is referred to as a "coordinate" and is displayed as two numbers separated by a comma. In this example: "14,68", the number "14" represents row "15" and "68" represents column "69". Therefore, "0,0" represents the upper left corner of the screen and "24,79" the bottom right corner of the screen.

The @...SAY Command

The command which accomodates the placement of prompts at alternate screen locations is:

```
@ {coordinates} [SAY {"prompt"}]
```

This command will position the prompt at the screen coordinates you specify.

Take a look at how it works. From the COMMAND LEVEL of the program, enter the following:

```
ERASE {CR}
@ 2,30 SAY "WHERE ARE YOU?" {CR}
@ 19,10 SAY "OVER HERE!" {CR}
@ 19,21 SAY "I FOUND YOU! {CR}
```

Displaying Text and Data at Alternate Screen Locations, cont.

The @ Command: Erasing Text

The "@...SAY" command can be used for erasing all or part of a line. To do so, enter the command followed by the coordinates of the *first* character of the line you want erased. Do not use the "SAY" option. When you press {CR}, it will erase that character *and every character which follows it on the line*.

Try it. Enter:

```
@ 2,30 {CR}
@ 19,15 {CR}
```

The @...SAY Command: Expressions

You have the option of using the "@" command in association with an "expression," as in:

```
@ {coordinates} [SAY {exp}]
```

This form is used to show the value of an expression comprised of one or more variables (data field or memory). For example, enter the following:

```
ERASE {CR}
USE People {CR}
@ 15,10 SAY Name {CR}
SKIP {CR}
@ 20,20 SAY Amount*2 {CR}
```

When you are finished, the name "Smith, Paul" and the amount "135.96" should appear at two different positions on the screen.

Note: If the variable/s you specify in the statement do not exist, an error will be returned.

Note: One form of the "@...SAY" command can be used to display the data in a specified format. That form is:

```
@ {coordinates} [SAY {exp}] [USING "format"]
```

Information on how to apply it is included near the end of the "Programming Tutorial".

The @...GET Command

The value (only) of a variable can also be displayed through the use of the "@...GET" statement:

```
@ {coordinates} [GET {var}]
```

This command will cause the program to place the contents of a data field or memory variable on the screen at the specified location. It can also be used in association with the "SAY" command as in:

```
@ {coordinates} [SAY {exp}] [GET {var}]
```

For example, type in the following from the COMMAND LEVEL of **H & D Base**:

```
ERASE {CR}
USE People {CR}
@ 14,10 GET Address {CR}
@ 15,10 SAY "ADDRESS: " GET Address {CR}
```

When you are finished, two addresses should appear on the screen -- one without a prompt and one with a prompt.

With the information you've been given to this point, it might seem that there is very little difference between the "@...SAY" and "@...GET" commands. There is, and the difference is a command called "READ".

The @...GET READ Command

The "@...GET" command should only be used in conjunction with the single-word command READ which may follow it anywhere in the COMMAND FILE. For example:

```
@ 14,10 GET NAME
READ
```

Displaying Text and Data at Alternate Screen Locations, cont.

When the READ command is encountered, the cursor reverts to the *first* "@...GET" statement in the COMMAND FILE and you are allowed to *alter* the information in that variable. With a {CR}, the new data is recorded, and the cursor is moved to the next "@...GET" statement you entered and the process is repeated (and so forth).

Note: A READ statement will only process "@...GET" statements which appear after any previous READ (or the beginning of the file).

Give it a try to see how it works. Enter the following *as a follow-up* to the last example we used:

```
READ {CR}
```

The cursor will revert to the beginning of the first address on the screen and you will be allowed to edit it. If you do not want to make any changes, simply press {CR}. Edit it if you like, and press {CR}. Now do the same thing on the second ADDRESS field. When you press {CR} this time, the information is saved to the file. Press {CR}, then type in the command "DISPLAY {CR}" to check your work.

Note: One of the following commands *must* appear after every 64 "@...GET" statements:

```
READ
ERASE
CLEAR GETS
```

You may use the "@...GET" and "READ" statements for *entering* as well as *editing* the information in variables. To do so, simply create a record *with no information in it*. When you do, the "@...GET" statement will pull up only the blank fields you specify allowing you to enter "new" information into them.

Displaying Text and Data at Alternate Screen Locations, cont.

The APPEND BLANK Command

The command for creating a record without any information in it is:

```
APPEND BLANK
```

When run, it creates a blank record, adds it to the bottom of the data file, and places the record pointer on it.

Here is a sample COMMAND FILE which demonstrates the techniques described above:

```
ERASE
USE People
APPEND BLANK
@ 10,10 GET Name
@ 11,10 GET Amount
READ
```

When this little program is run, it will add a record to the bottom of our sample database file named "People," with user-specified information in the "Name" and "Amount" fields. The rest of the fields will be blank.

The @...GET... PICTURE Command

The "@...GET" command can be used in one final form:

```
@ {coordinates} [SAY {exp}] [GET {var}]
    [PICTURE {format}]
```

Using this form of the "@...GET" statement, you can create a *template* which will be used when the data is displayed.

The format may take a form something like this:

```
@ 10,5 SAY "Date: " GET DATE PICTURE "99/99/99"
```

Assuming that the "Date" field is blank, it will appear on the screen as:

```
Date:  /  / :
```

Displaying Text and Data at Alternate Screen Locations, cont.

This format will allow up to six *numbers* to be entered, and the slashes (/) cannot be moved or erased. When stored in the field of a file, the slashes *will* appear.

Note: Data field variables must be large enough to include the data *and* any additional symbols included in the template or the data will be truncated.

These five characters may be used in defining the types of information which will be accepted by "PICTURE" statements:

1.) 9 :Allows only numbers to be entered
2.) # :(Same as #1)
3.) A :Allows only alphabetic characters to be entered
4.) X :Allows *any* character to be input
5.) ! :Converts any character input to uppercase

Here are some samples of the ways some of these could be applied and what their output would appear like:

```
1.) COMMAND: @ 1,1 SAY "PHONE NUMBER" GET
             MPhone PICTURE "(999)999-9999"
    DISPLAY: PHONE NUMBER:(   )   -    :
      INPUT: Only numbers can be entered

2.) COMMAND: @ 1,1 SAY "LAST NAME" GET Mname PIC-
             TURE "AAAAAAAAAAAAAAAAAAAA"
    DISPLAY: LAST NAME:                  :
      INPUT: Only alpha characters can be entered

3.) COMMAND: @ 1,1 SAY "NAME" GET MName PICTURE
             "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
    DISPLAY: NAME:                            :
      INPUT: Any alpha character input will be forced to
             upper case
```

You may put other characters in the template if you like, but they will be interpreted literally (as with the slashes used above).

The "@...SAY...USING" command is very similiar to the "@...GET...PICTURE" statement. Consult the "Reference" section for further information.

Hopefully, it is becoming obvious to you how all of these commands fit together. The primary use of the "@...SAY" command is to put titles and prompts on the screen. The "@...GET" statement is used to enter or alter information. Together they form the basic elements of construction for program menus and screens.

Saving an @...GET Format

Any COMMAND FILE which contains *only* "@...SAY...GET" commands and comments (preceded by asterisks) can be saved (with {CTRL-W}) and used as a template for future interaction with a database. This allows you to override the standard form to which the program defaults when in the APPEND, EDIT, and INSERT modes and create custom screens for those purposes. For example, you can display special prompts, and allow users to interact only with data in specified fields of a record.

All COMMAND FILES are stored on the disk with a ".CMD" trailer. In order to use a COMMAND FILE in the manner described above, you *must* change that trailer to ".FMT" to signify that it is to be used as a Format File. Use the RENAME command to do so.

When you want to use your custom data template instead of the standard one supplied by the system, enter the following from the COMMAND LEVEL of **H & D Base** (or include it in a COMMAND FILE):

```
SET FORMAT TO {filename}
```

With this command entered, the format detailed in the {filename} you specified will be used in the APPEND, EDIT, AND INSERT modes.

Displaying Text and Data at Atlernate Printer Locations

You learned earlier in this manual that the command "SET PRINT ON" sends information to the printer *as well as the screen*. The "SET FORMAT TO PRINT" command sends information to the printer *instead of* the screen. If you had instituted either of them before entering the examples in this "Programming Tutorial," you would now have a number of pages of print with text and data scattered about them.

You may format the appearance of information on your printer much like you did for that which appeared on the screen. With this ability, you can set up files to print checks, invoices, purchases orders, and other standard forms.

The @...SAY...USING Command

Use the "@...SAY...USING" command to format information being sent to the printer. In its full form it reads:

```
@ {coordinates} SAY {var/exp/"string"} [USING {"format"}]
```

It is applied exactly as if the output were being sent to the screen. You have the option to print the current value of a variable, the result of an expression, or a literal string prompt message.

The USING portion of the statement is much like the PICTURE command (discussed previously), except that the exclamation point (!), which converts all lower case letters to uppercase, cannot be used.

There are a few limitations regarding printer output that you should be aware of:

1.) GET and PICTURE phrases should not be used.

Displaying Text and Data at Alternate Printer Locations, cont.

2.) READ commands cannot be used.

3.) Coordinates *must* be in order. A statement which sends information to line #19 must appear before a statement which sends information to line #20. The same holds true for information sent to any single line.

4.) Do not use the "SET PRINT ON" statement as a replacement for "SET FORMAT TO PRINT".

The "coordinates" of your particular printer will not, more than likely, match those of your screen. Some units print 10 characters per inch (horizontally), while others do 12. On the other hand, some printers print 6 lines per inch (vertically), while others will include 8.

Find out how many characters per inch (horizontally and vertically) your machine prints (in your printer's manual). Then measure from the left side and top of the paper where you want a particular piece of information to appear. Multiply the width measurement by the number of characters per line your machine prints to get the "Column" coordinate, and the heighth measurement by the number of lines per inch your machine prints to get the "Row" coordinate.

Note: Neither the "Row" nor the "Column" coordinates can be larger than 255.

Here is just a section of a COMMAND FILE which uses information from a database to fill-in the fields of a check on the printer:

```
SET FORMAT TO PRINT
@ 8,38 SAY Date
@ 12,10 SAY Payee
@ 12,65 SAY Amount
@ 13,10 SAY Address
@ 14,10 SAY City
@ 14,25 SAY State
@ 14,29 SAY Zip
```

Displaying Text and Data at Alternate Printer Locations, cont.

The EJECT Command

The EJECT command causes the printer to do what is called a "form feed" -- a fancy term for ejecting the page and repositioning itself at the top of the next one. This can only be accomplished if the printer has already been activated with a "SET PRINT ON" or "SET FORMAT TO" command.

The length of the form ejected is set through a dial which appears somewhere on your printer. It should be set to match the length of the form you are using.

If you are printing a number of forms, filling each with information supplied by a different record, put your "@...SAY" commands in a loop with the EJECT command at the bottom. When run, it will eject the form in the printer, reposition itself at the top of the next form, and set the coordinates back to "0,0".

As applied to our above example, a loop of this type would read:

```
USE Accounts
SET FORMAT TO PRINT
DO WHILE .NOT. EOF
    @ 8,38 SAY Date
    @ 12,10 SAY Payee
    @ 12,65 SAY Amount
    @ 13,10 SAY Address
    @ 14,10 SAY City
    @ 14,25 SAY State
    @ 14,29 SAY Zip
    SKIP
    EJECT
ENDDO
```

If you attempt to print information beginning at coordinates which *come before* those of the last statement run, an *automatic* EJECT will be issued.

Using Macros In a Command File

The Ampersand Symbol

Whenever an ampersand (&) followed by the name of a character string memory variable is encountered in a command included in a COMMAND FILE (only), **H & D Base** replaces the ampersand and memory variable name with the memory variable's character string. This is referred to as "Macro Substitution."

One of the most useful applications of this "pseudo-function" is in the handling of complex expressions which are used over-and-over again in a program. The expression can be stored in a string memory variable and called back at will with the ampersand followed by the name of the variable.

For example, here is a commonly-used routine for rounding numbers:

```
INT((Mnum + .005)*100)/100
```

It can be stored to the memory variable "Mround" like this:

```
STORE "INT((Mnum + .005)*100)/100" TO Mround
```

and brought back in a program like this:

```
STORE 65/3 TO Mnum
DISPLAY &Mround
```

This small program would divide 65 by 3, then place a rounded version of the answer on the screen.

Another common use of "Macro Substitution" is in association with a FIND command in sequences where a string is entered into a variable then located within a database.

Macros can be used only with string memory variables (not data field variables), in COMMAND FILES *only*.

Leaving a Command File

Once the last command in a COMMAND FILE has been read and acted upon, you are automatically returned to the COMMAND LEVEL of the program (assuming that the COMMAND FILE was not nested). At times, you will want to "break out" of a COMMAND FILE *before* you reach that last command.

There are a number of statements you may use. If the COMMAND FILE you are "breaking out of" is *not* "nested," the RETURN command (covered above) *will* deposit you at the COMMAND LEVEL of the program.

The CANCEL Command

To return to the COMMAND LEVEL of **H & D Base** from *any* COMMAND FILE (nested or otherwise) at *any* time, use the command:

```
CANCEL
```

The QUIT Command

To return to the Atari DESKTOP from *any* COMMAND FILE, include this command in your program:

```
QUIT
```

Both the CANCEL and QUIT statements must be included in the body of a program and cannot be issued from the keyboard during the "run" of a COMMAND FILE.

The ESCAPE Key

To terminate the execution of a COMMAND FILE *from the keyboard* during its "run," press the "Escape Key" {ESC}. You will be returned to the COMMAND LEVEL of the program.

Printing a Command File

One **H & D Base** command has been included expressly for the purpose of generating a print-out of a COMMAND FILE. That command is:

```
DISPLAY COMMAND {command file name}
```

When entered from the COMMAND LEVEL of the program, this statement will print out a copy of the designated COMMAND FILE on the printer. Make sure the printer is set for printing *before* attempting to execute this command.

You *may* substitute the word LIST for DISPLAY.

Note: You may list only files with the ".CMD" trailer using this command.

Programming — Helpful Hints

You may find the following points useful as you design and code your **H & D Base** COMMAND FILES:

1.) Plan the whole file out on paper *before you begin*. Determine what the problem is that you want to solve with the program first, then define the steps (in order) that need to be taken to solve that problem.

2.) As you enter the actual code, break up specific tasks into sections and work on one section at a time.

3.) Talk to yourself. Use the NOTE (or *) statement frequently within a program to indicate what is happening at a particular point.

4.) Indent lines of code freely as an aid for determining the flow of the program.

5.) As an aid to the debugging of your programs, place WAIT statements at the end of routines. Execution of a program will pause when the statement is reached allowing you to more easily to determine which routine it is that is causing an error.

6.) Don't take the easy way out. Free use of commands like CANCEL are "cop-outs". Program to a logical conclusion.

7.) Provide as many error traps as you possibly can. Never over-estimate the intelligence of the people who may be using your programs.

8.) When run, a question mark (?) appearing on a line by itself in a COMMAND FILE will generate a blank line on the screen. Use it for spacing-out displays.

Programming — Helpful Hints, cont.

9.) Use the ERASE command frequently. Seldom does it hurt to make sure the screen is clear.

10.) As you write your program code, you might find it helpful to enter all commands in upper case letters, and all file names, variable names, etc. in lower case letters.

Summary

This concludes the *Programming Tutorial* portion of the manual. It has been our purpose in this section to acquaint you with the nature of **H & D Base** COMMAND FILES, and most of the statements which are commonly used in conjunction with them.

Please understand that we have *not* discussed every single form that each of the commands can take. In addition, there are a number of commands which have not been covered *at all* in this, or any other part of the **H & D Base** tutorial. Most of them are so specialized that you will have little cause to *ever* use them. That does not diminish, however, the importance of you continuing your studies from this point with a careful examination of the "Reference" section of this manual.

We have included a COMMAND FILE on your Program Disk which is fully operable and contains many of the statements we've covered in this section. It may be useful for you to examine it carefully.

We do hope that you enjoy the time you spend programming in **H & D Base**, and that the hours you put in now will render solutions which will save you many more in the future.

Chapter 7
REFERENCE

Introduction

This is the *Reference* section of the **H & D Base** manual. In it you will find concise procedural listings for all of the program's commands, functions, and operations. A number of examples are also included.

The *Reference* is not intended to be a substitute for the four tutorial lessons which precede it. A thorough study of that section is *mandatory* for a proper understanding of the **H & D Base** program and how it operates.

Use this *Reference* section at those times when you can't quite remember how a tutorial lesson instructed you to accomplish a certain task. It will also be handy to check on alternate command forms that have not been previously covered.

The *Reference* begins with an overview of the symbols which are used extensively in this section. It continues (and concludes) with a discussion of each command (in alphabetic order). The miscellaneous functions and operators of **H & D Base** are included in that alphabetic listing, but are grouped (respectively) under the titles "FUNCTIONS" and "OPERATORS".

Additional information can be found in the APPENDICES portion of this manual.

We trust that you will find this *Reference* section both handy and helpful.

Symbol Definitions

Many of the commands reviewed in the *Reference* section which follows include special symbols as part of their formats. In order to make use of the full potential of **H & D Base**, it is important for you to understand the meaning of these symbols. They are listed alphabetically.

{**command**} **or** {**statement**}	A {command} or {statement} is any valid **H & D Base** instruction. Examples: REPORT, INDEX, IF...ENDIF
{**char string**} **or** {**cstring**} **or** {**string**}	These statements stand for "Character String." A character string is *any* character or characters which are enclosed in matching quotes (single or double). Examples: "ABCD", '1!2@3#4$5%'
{**delimiter**}	A {delimiter} is a special keyboard character which is used to punctuate information. There are three valid {delimiters}. They are: commas (,), single quotes ('), and double quotes ("). Example: Doe,John,234 Maple,Akron,OH,43567,758-8765
{**exp**}	Stands for "expression." An expression can be composed of numbers, functions, field names or character strings. They must be listed in a prescribed manner. (See the *Intermediate Command Tutorial* for further information. Example: $('barney' + &MLast,n,10)
{**exp list**}	Stands for "expression list." An expression list is a series of valid expressions separated by commas. Example: (4 + 1),(M10),(MLast = "Smith")
{**field list**} **or** {**list**}	A "field list" (or "list") is a series of record field names separated by commas. Example: Name,Address,City,State,Zip

{file} or {file name}	Either of these symbols stands for any valid **H & D Base** file name. Rules for the creation of valid file names are included in the first section of the *Beginning Command Tutorial.* Examples: NAMES or INVNTORY
{form file}	{form file} is used to represent the name of a report form file of the type ".FRM". For further information consult the "REPORT" command in the *Reference* section. Example: Salesrep.FRM
{index file}	An {index file} of the type ".NDX" is created when the INDEX command is executed in relation to a "Key" field (or fields) in the USE file. For further information, consult the "INDEX" command in the *Reference* section. Example: Zipind.NDX
{key}	A {key} is the field/s or expression/s that a file is indexed upon. There may be several indexes for any given database, each using different {keys}. Example: INDEX ON Zipcode TO Zipind ("Zipcode" is the {key})
{memvar}	{memvar} represents any "memory variable." Memory variables represent values which may be changed, and are stored in a reserved area of the computer's memory. Example: STORE "Smith" TO MLast ("MLast" is a memory variable)
{memvar list}	Stands for a list of memory variables (separated by commas). Example: Mem1,Mem2,Mem3
{*n*}	*n* represents a literal number. They cannot be derived from calculations or obtained from memory variables. Examples: 2 or 1234567890 Invalid: ABCDE or (2 + 1234567890)
{scope}	{scope} refers to the range or extent of action that a command may take. There are five valid forms represented by the {scope} symbol:

Symbol Definitions, cont.

1.) ALL - Every record in the file
2.) NEXT *n* - The next *n* records in the file including the current record (*n* must be a literal value)
3.) RECORD *n* - Only record *n* (*n* must be a literal value)
4.) FOR {exp} - Any record as long as the {exp} returns a "True" value
5.) WHILE {exp} - Any record *until* the first record which returns a "False" value to the {exp}

{skeleton}

{skeletons} are abbreviated or wild card representations of file names (or groups of file names). They can also be used in the same manner with memory variables (or groups of memory variables).

Two characters may be substituted for characters in the name/s of files and memory variables. They are the question mark (?) and the asterisk (*). A question mark in a filename (including the "trailer") or {memvar} indicates that any character can occupy that position.

Examples:
```
Names.DAT = N????.DAT
Names1.DAT and Names2.DAT = Names?.DAT
Temp.DAT and Temp.NDX = Temp.???
```

An asterisk indicates that *any* character can occupy that position *plus* all the positions in the filename before the asterisk (to the beginning of the name), after the asterisk (to the end of the name), or between the asterisk and the next listed character/s.

Examples:
```
*.DAT = Names.DAT and Ledger.DAT
Names.* = Names.DAT and Names.NDX
*.* = Names.DAT and Ledger.NDX
N*.DAT = Names.DAT and Numbers.DAT
N*.* = Names.DAT and Numbers.NDX
```

Any symbol not covered in this section is particular to a specific command and will be discussed along with the rest of the material on that command.

?

FULL FORM: ? [{exp list}]
?? [{exp list}]

PURPOSE: Used to show the value of an expression or list of expressions.

OVERVIEW: The "?" and "??" commands are specialized forms of the DISPLAY command (equivalent to "DISPLAY OFF {exp}"). They can be used in association with memory variables, database fields, constants, or functions.

If *no* expression is specified, the "?" will produce a blank line when outputted.

The double question mark command ("??") behaves like a single question mark command ("?") except that no line feed and carriage return is done before the expression is printed. This can be used in command files to output more than one expression to the same output line.

If a SET RAW ON command is in effect, then no spaces will be placed between items in a list; otherwise one space separates items.

EXAMPLE #1: Various applications of the question mark command:

```
? 2 + 2
4 ok
USE  Names
ok
DISPLAY City
Miami ok
? City
Miami ok
```

&

FULL FORM: &{cstring memvar}

PURPOSE: Used for placing the stored value of a character string memory variable in a command

OVERVIEW: Whenever an ampersand (&) followed by the name of a character string memory variable is encountered in a command, **H & D Base** *replaces* the ampersand *and* the memory variable with the value of that variable. Note: Ampersands can be used in COMMAND FILES only.

The name of the variable must appear *immediately* after the ampersand (no spaces).

EXAMPLE #1: Substituting an ampersand for a character string memory variable:

```
STORE "Frank" TO MFirst ok
USE People ok
INDEX on Name TO Nameind ok
00005 RECORD(S) INDEXED ok
Use People INDEX Nameind ok
FIND &MFirst ok
DISPLAY Name

00001 Frank Williams ok
```

If you desire to append characters to the value of the variable when it is displayed you may do so. If they are to appear *immediately following* the value of the variable a period (.) must separate them. In all other cases, simply use one or more blank spaces.

EXAMPLE #2: A substitution with additional characters:

```
STORE "B:" TO MDrive ok
USE &MDrive.Names ok
```

These commands would USE a file called "Names" which is located on the "B" drive.

If an ampersand is *not* followed by a valid memory variable name, no expansion is attempted and the ampersand remains in the command line.

@

FULL FORM: @ {coordinates} [SAY {exp} [USING {format}]]
[GET {variable} [PICTURE {format}]

PURPOSE: Used to format console screen or printer output

OVERVIEW: This the most powerful command available for displaying specific, formatted information on the screen or the printer. It can take a variety of different forms, depending upon how it is used in conjunction with the SET FORMAT TO, ERASE, EJECT, CLEAR, USING, and GET statements. All of the combinations are discussed below.

{Coordinates} are screen or printer "intersections" and are represented by numbers for a particular "row" and a particular "column" in this manner: "row,column".

The Atari monitor coordinates have a "row" range of 0-24, and a "column" range of 0-79 (25 rows by 80 columns). Printers ordinarily have both a "row" *and* a "column" range of 0-254. A coordinate pair of 0,0 represents, therefore, the first character location on the upper left corner of the display or printer. The pair 20,30 represents the 21st line and the 31st column -- and so forth.

Coordinates may also be numeric memory variables and numeric expressions in addition to literals (as above), as long as they are integer values.

The SET FORMAT command is used to specify whether output is sent to the screen or the printer. Note: SET FORMAT TO SCREEN is the default.

When the FORMAT has been SET TO the SCREEN, the "@" command causes data to be displayed *on the screen*. "@" commands may be issued in any order *to the screen*. For example you can "SAY" something to row 23 *before* you "SAY" something to row 5. The same is true for column designations (as long as they are being displayed *on the screen*).

When a SET FORMAT TO PRINT command has been issued, the "@" command causes data to be printed on the printer. "@" commands to the printer *must* be output in order; e.g., output to row 5 *must* be issued before any that is sent to a row with a larger number (like row 23). The same is true for column designations. If an attempt is made to issue information to a row or column with a number lower than that previously displayed (on that page), the printer will eject a full sheet of paper before printing it.

When in the SET FORMAT TO SCREEN mode, an ERASE command will clear the screen of all information that was previously on it, as well as release all the GETs (see below), and reset the coordinates to "0,0". When in the SET FORMAT TO PRINT mode, an EJECT command issues a page feed and resets the coordinates to "0,0".

The SAY phrase is used to display an expression which is not to be altered by subsequent editing (such as a prompt). The optional USING sub-phrase (see below) is used to format the expression emitted by the SAY phrase. SAY phrases may be used on either the screen or the printer.

The GET phrase displays the current value of an *existing* field variable or memory variable. As opposed to a SAY, this value *is* subject to alteration (through the READ command). The optional PICTURE sub-phrase (see below) may be used with a GET phrase to allow special formatting and validation of the data as it is entered (see the READ command for further information). If no PICTURE clause is given, the data type (character, numeric or logical) forms an implicit PICTURE.

If the data type of the field variable or memory variable in the GET is logical, the data validation allows only the characters 'T', 'F', 'Y', 'N', and their lower case equivalents to be entered.

A maximum of 64 GETS can be active at any given time. Either the ERASE command or the CLEAR GETS command may be used to release the existing GETS.

GETS are only recognized when the SET FORMAT TO SCREEN command has been issued.

A READ command must be issued in order to “fill” the GETS with the desired information (SET FORMAT TO SCREEN only). Normally, a number of “@” commands are issued along with GET phrases and a READ command is issued to allow editing or data entry into the GET variables.

EXAMPLE #1: The proper structure for using the READ command in association with @...GET statements:

With a database:

```
USE Names
APPEND BLANK
@ 10,15 SAY "Name " GET Name
@ 12,15 SAY "Address " GET Address
@ 14,15 SAY "City " GET City
@ 14,40 SAY "State " GET State PICTURE "!!"
@ 14,52 SAY "Zip " GET Zip PICTURE "99999"
@ 16,15 SAY "Account Number " GET acct:num PICT "99999" READ
```

The USING and PICTURE options are templates which can be used to indicate which characters are to appear on the screen or page (and what form they are to appear in). USING can only be used in relation to the SAY command, and PICTURE can only be used in relation to the GET command.

The following table defines the characters (and their functions) which can be used in these templates:

FORMAT CHAR.	“SAY...USING” FUNCTION	‘GET...PICTURE” FUNCTION
# or 9	Causes the next number to be output	Allows only a digit (1,2,...,8,9,0) and the characters “.”, “ + ”, “-”, and “ ” (a space) to be entered
X	Outputs the next character	Allows any character to be entered
A	Outputs the next character	Allows only alphabetic characters to be entered
$ or *	Outputs either a digit or a $ or * instead of leading zeros	No effect
!	No effect	Converts lowercase alpha characters to uppercase

Any character which does not appear in this chart is "non-functional" and will be inserted into the variable if used.

EXAMPLE #2: A sample @...SAY...GET...PICTURE statement:

```
@ 10,1 SAY 'ENTER DATE: MM/DD/YY' GET MDate PICTURE '99/99/99'
READ
```

The message 'ENTER DATE: MM/DD/YY' is displayed, followed by " / / ", (assuming that there was no value to the variable "MDate" prior to issuance. When the READ command is issued, the program will allow only digits to be entered (example: 04/25/88).

EXAMPLE #3: A sample @...SAY...USING statement with a "Floating Dollar" template:

```
@ 18,45 SAY MHours*MRate USING '$$$$$.99'
```

This "@" command could be used with either the screen or the printer since it has no GET phrase. It could be used to print the amount on a set of payroll checks. The dollar signs will be printed as long as there are leading zeros in the value to be printed.

If "MHours" = 10, and "MRate" = 5.00, then '$$$50.00' will be displayed. This is the "Floating Dollar" feature and is valuable for printing checks whose values cannot be easily altered.

Commas may be used in the integer part of a picture. If there are no digits in the value to the left of where they appear, they will be replaced by the picture character in front of them.

EXAMPLE #4: Using memory variables as coordinates:

```
SET FORMAT TO PRINT
GO TO TOP
STORE 7 TO MCount
DO WHILE .NOT. EOF
  IF MCount = 50
    EJECT
    STORE 7 TO MCount
  ENDIF
  @ MCount,12 SAY Name USING 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  @ MCount,48 SAY Address USING 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  @ MCount,64 SAY City USING 'XXXXXXXXXXXXXXXXXX'
  @ MCount,88 SAY State USING 'XX'
```

@, cont.

```
    @ MCount,104 SAY Zip USING 'XXXXX'
    STORE MCount + 1 TO MCount
    SKIP
  ENDDO
  RETURN
```

This COMMAND FILE, when run, will print a record on a single line then skip to the next record and do the same until the end of the file is reached.

When the FORMAT has been SET TO the SCREEN and neither a SAY or a GET phrase is used, then the remainder of the line indicated by the coordinates is erased. For example, the command "@ 22,0" will clear the entire 22nd line.

The last form of the SET FORMAT command is: SET FORMAT TO {format file}. When it is in effect and a READ command has been issued, the "@" commands are READ from the predesigned {format file}. In this manner, the user may design the screen into a format for more specialized purposes. Note: The use of format files is not necessary for use of "@"s, since "@"s may reside in COMMAND FILES. See READ for more information.

Accept

FULL FORM: ACCEPT ["{string}"] TO {memvar}

PURPOSE: Prompts user to input a character string into a specified memory variable. Does not necessitate the use of quotes.

OVERVIEW: This command works very much like the INPUT command in that it facilitates the entry of character strings into memory variables from the keyboard. The main difference lies in the fact that the string does *not* have to be entered with quotation marks around it. ACCEPT makes a "CHARACTER" memory variable out of *whatever* is entered.

The command will create the memory variable and store the input to it.

If a prompt ("{string}") is included in the phrase, it is displayed on the screen. You may use single quotes or double quotes to delimit the prompt string, as long as both the beginning and the ending delimiters correspond.

If a carriage return is entered in response to an ACCEPT request, a single blank space will be placed in the specified memory variable.

EXAMPLE #1: An ACCEPT command with a prompt:

```
ACCEPT "ENTER COMPANY NAME" TO MName
ENTER COMPANY NAME Ajax ok
DISPLAY MEMORY
MName          (C)         Ajax
**TOTAL**      01 VARIABLES USED      00005 BYTES USED ok
```

Append

FULL FORMS:

1.) APPEND
2.) APPEND BLANK
3.) APPEND FROM {file} [FOR {exp}] [SDF] [DELIMITED] [WHILE {exp}]

PURPOSE: Used to add records to a database

OVERVIEW: The APPEND command, in all three of the forms listed above, adds records to the "bottom" of the database in USE. The only other command which can be used to add records to a database is INSERT, a command which places but one record at a time at a specified position.

The first form of the APPEND command, in which there are no qualifying clauses, allows users to enter data opposite each field name in the USE file's structure. Any number of new records may be created from the keyboard in such a manner. The APPEND mode is terminated when a carriage return is entered as the first character of the first field of a record.

EXAMPLE #1: A simple APPEND:

```
USE People ok
DISPLAY STRU

STRUCTURE FOR PEOPLE.DAT
INDEX IN USE: NONE
NUMBER OF RECORDS: 5
PRIMARY SELECTED

FIELD     NAME        TYPE     LENGTH      DEC
01        NAME         C        030
02        ADDRESS      C        030
03        CITY         C        020
04        STATE        C        002
05        ZIP          C        005
06        AMOUNT       N        007         2
```

```
TOTAL BYTES:                         00095  ok

APPEND

RECORD 00006
NAME:      Howard, Roy
ADDRESS:   6587 N. Benton
CITY:      Chicago
STATE:     IL
ZIP:       48967
AMOUNT:    345.76

RECORD 00007
NAME: {CR} ok
```

If the database in USE is an indexed database and the index file is in current use, then the index file/s specified in the USE command is automatically updated when the new records are appended (except for APPEND BLANKS). Any index file associated with that database which is *not* in use when the database is APPENDED must be re-indexed.

If the SET CARRY command is activated (turned "ON"), all of the data from a record will be carried over to the next record. Changes can then be made in the new record. This is extremely useful if successive records share a great deal of common data.

If a SET FORMAT TO {format file} is in effect, APPEND will use the "@" commands from the specified format file to format the screen and restrict access (if desired) to certain fields in the record.

The second form of the APPEND command is APPEND BLANK. When this command is issued, one record, filled *completely with blanks* is appended to the USE file. This record can then be filled by the EDIT, REPLACE, or READ statements.

The third form of the APPEND command allows you to add records to a file *from a completely different file*. The {file} can be another **H & D Base** file, a "System Data Format" file (SDF), or a file whose contents have been separated with a "delimiter."

If the optional SDF or DELIMITED clauses are *not* used, the FROM file is assumed to be a standard **H & D Base** database file. When executed, the structures of the USE and FROM files

Append, cont.

are compared and fields which occur in the records of *both* files are taken from the FROM file and appended to the USE file. Padding and truncation are performed as appropriate to force the FROM data items into the USE file's structure.

If the optional FOR phrase is also used, the program will append only the records of the FROM file for which a certain condition {exp} is true. The procedure will continue until the end of the FROM file is reached. The fields used in the expression must reside in the file receiving the new records.

EXAMPLE #2: This example demonstrates an APPEND FROM {file} FOR {exp}:

```
USE Names
LIST Zip,Acct:Num

00001     FL   10235
00002     IL   76926
00003     AZ   58938
00004     WA   38769
00005     TX   02678
00006     AZ   58397
00007     CA   10003
00008     NV   20835
00009     CO   69361
00010     MA   48376 ok

COPY STRU TO Temp ok
USE Temp ok
APPEND FROM Names FOR State = "AZ" .AND. Acct:Num > 58500 ok

LIST

00001   MURPHY, VINCE   987-B Hallowell
Phoenix   AZ   87356   58938
```

If the SDF clause is used, the records to be transferred are assumed to be in "System Data Format" (see *Appendix B*).

EXAMPLE #3: APPENDING with the information from a "System Data Format" file:

```
USE Names ok
APPEND ALL FROM Test.TXT SDF ok
```

Many computer languages generate files where character strings are enclosed in delimiters (usually single or double quotes) and fields are separated by commas. These files are referred to as "delimited." If the DELIMITED clause is used along with the APPEND command, the records taken from the FROM file are assumed to be delimited and are appended accordingly. The program removes the delimiters and commas from them and stores the data in a structure standard to **H & D Base**.

For further information on the transfer of SDF and DELIMITED files, consult *Appendix B.*

The APPEND command is especially useful when it is necessary to expand/contract fields or add/delete fields from an existing database. Using the CREATE command, set up a new database containing the desired structure and then APPEND the old database to the new. Fields which appear only in the new database will be filled with blanks.

Browse

FULL FORM: BROWSE [FIELDS {field list}]

PURPOSE: Used to view and/or edit a database file in a full screen "window" format

OVERVIEW: When the BROWSE command is executed, the data from up to 20 records is displayed on the screen -- one line per record. If the total number of characters in the fields of the records are greater than 80, part of the data will be beyond the right edge of the screen. The screen should, therefore, be considered as a "window" into a database. You can scroll backwards and forwards through the records and you can pan left and right through the fields of the database. Any data can be edited with the standard full-screen editing method.

If no {field list} is supplied, BROWSE will show all fields in the same order as the structure.

These are the control keys which will work with the BROWSE command:

1.) {UP ARROW} or {CTRL-E}
Moves the cursor one field to the left
2.) {DOWN ARROW} or {CTRL-X}
Moves the cursor one field to the right
3.) {LEFT ARROW} or {CTRL-S}
Moves the cursor one character to the left
4.) {RIGHT ARROW} or {CTRL-D}
Moves the cursor one character to the right
5.) {HOME}
Moves the cursor to the first character in the record
6.) {CTRL-B}
Scrolls the screen one *field* to the right

7.) {CTRL-Z}
Scrolls the screen one *field* to the left

8.) {CTRL-N}
Inserts a line (for a new record)

9.) {CTRL-G} or {DELETE}
Delete character under cursor

10.) {BACKSPACE}
Delete character to left of cursor

11.) {CTRL-U}
Delete record toggle

12.) {CTRL-A} or {CTRL-R}
Save record - Back up to previous record

13.) {CTRL-F} OR {CTRL-C}
Save record - Advance to next record

14.) {CTRL-W}
Exit - Save changes to current record

15.) {CTRL-Q}
Exit - Do *not* save changes to current record

Special Note: Atari has chosen to designate {CTRL-C} as the code for exiting from a program and returning to the DESKTOP. It is "hard wired," but can be overridden (to a certain degree) by software programmers. If you press {CTRL-C} two or more times *in rapid succession*, you will automatically exit **H & D Base** and return to the DESKTOP. We have included {CTRL-C} only as a convenience for those who use it extensively in other programs.

Cancel

FULL FORM: CANCEL

PURPOSE: Used to halt the execution of a COMMAND FILE

OVERVIEW: The CANCEL command can only be used in a COMMAND FILE. It is used to bring a halt to all COMMAND FILE operations. Control returns to the COMMAND LEVEL of the program when it is encountered.

EXAMPLE #1: The use of CANCEL in a COMMAND FILE:

```
DISPLAY "ARE YOU READY TO QUIT?"
INPUT "ENTER Y OR N " TO X
IF X
   CANCEL
ENDIF
```

The INPUT command is asking for a "Yes" or "No" answer. If the response is "Yes" ('Y', 'y', 'T', or 't'), then the "IF X" line of the command file will be satisfied (since "X" will be logically .TRUE.) and the CANCEL command will be executed.

Clear

FULL FORM: CLEAR [GETS]

PURPOSE: Used for resetting the system

OVERVIEW: When the CLEAR command is issued from either the COMMAND or the PROGRAM LEVEL of **H & D Base**, *everything* is completely reset. All databases in USE are closed and deselected, all memory variables are released, and the PRIMARY work area is re-selected.

If the optional "GET" (or "GETS") statement is used in association with the CLEAR, all of the GETS which have been set up by the "@" command are cleared. The screen is left *as is*. Note: The "ERASE" command will accomplish the same purpose with one notable differance: The screen *is* erased.

It is a good programming practice to include a CLEAR command at the beginning of all COMMAND FILES to insure a "clean slate" (unless the COMMAND FILES are "nested").

Continue

FULL FORM: CONTINUE

PURPOSE: Used to resume the execution of a LOCATE command

OVERVIEW: The CONTINUE command can only be used in association with the LOCATE command. When encountered, it resumes the LOCATE operation. See LOCATE for more information.

Copy

FULL FORMS:

1.) COPY [{scope}] TO {file} [FIELD {field list}] [FOR {exp}] [SDF] [DELIMITED [WITH {delimiter}]] [WHILE {exp}]

2.) COPY STRUCTURE TO {file} [EXTENDED] [FIELD {fieldlist}]

PURPOSE:

Used for copying the records (all or part) or structure (all or part) of the database in USE to another file

OVERVIEW:

You may specify the {scope} of the records to be copied (default = all records). The destination {file} specified in the command is created if it does not exist. If a file already exists with the same name, *all information in that file is destroyed*.

If the optional FIELD clause *is not* used, all fields will be copied.

EXAMPLE #1:

A simple COPY:

```
USE Names ok
COPY ALL TO Temp
00010 RECORDS COPIED ok
USE Temp ok
LIST Name,Acct:Num

00001  HOWARD, ROY        10235
00002  THOMAS, BRENDA     76926
00003  MURPHY, VINCE      58938
00004  ALIMBO, JENNY      38769
00005  HAMILTON, JOHN     02678
00006  HORMAN, EILEEN     58397
00007  CUCUK, CHERYL      10003
00008  ZACHRY, DAVE       20835
00009  FAORO, ASHLEY      69361
00010  MADDEN, KRIS       48376 ok
```

If the optional FIELD clause *is* used, the name of a field (or list of fields) must be supplied. Only the specified fields will be copied to the destination {file}. The new structure will be made up of only those fields specified by the FIELD clause.

Copy, cont.

EXAMPLE #2: A COPY in which only specified fields are transferred:

```
USE Names ok
COPY ALL TO Temp FIELDS Name,City,State
00010 RECORDS COPIED ok
USE Temp ok
DISPLAY STRU

STRUCTURE FOR TEMP.DAT
INDEX IN USE: NONE
NUMBER OF RECORDS: 10
PRIMARY SELECTED

FIELD       NAME         TYPE       LENGTH        DEC
01          NAME           C        030
02          CITY           C        015
03          STATE          C        002

TOTAL BYTES:                        00048   ok
```

Three fields of the ten records were copied to "Temp" from "Names".

If the SDF clause is specified, the file in USE is copied to the destination file in *System Data Format*. This is a format which can be used (or adapted for use) by a number of data processors other than **H & D Base**.

If the optional DELIMITED keyword is included in the command, the destination file will have all of its character string type fields enclosed in double quotes (no quotes for numeric fields) unless an alternate delimiter is specified using the "WITH" subphrase. The fields will be separated by commas. Note: This is the converse action to a delimited APPEND.

EXAMPLE #3: Transferring information to a file DELIMITED with single quotes and commas:

```
USE Names ok
COPY RECORD 8 TO Temp FIELDS Name,City,State,Acct:Num DELIMITED
8 RECORD(S) COPIED ok
```

This is what the new record #1 (old record #8) looks like DELIMITED:

```
"CUCUK, CHERYL","San Jose","CA","94687",10003
```

EXAMPLE #5: COPYING information to a file DELIMITED by characters other than double quotes:

```
USE Names ok
COPY ALL TO Temp DELI WITH @ ok
```

If you use either the DELIMITED or SDF options, the destination {file} will be created with a ".TXT" trailer. In all other cases, the destination {file} will have a ".DAT" trailer.

In the second form of the command, the one which includes the "STRUCTURE" statement, only the structure of the **H & D Base** file in USE is copied to the destination file. The structure of the destination will include only the fields you specify if you choose to append "FIELD {fieldlist}" to the COPY STRUCTURE phrase.

The COPY STRUCTURE EXTENDED command will copy the structure of the database in USE to the designated {file} *as records*. This allows you to examine the structure during the execution of a program.

EXAMPLE #4: The EXTENSION of a database file structure:

```
USE People ok
DISPLAY STRU

STRUCTURE FOR PEOPLE.DAT
INDEX IN USE: NONE
NUMBER OF RECORDS: 5
PRIMARY SELECTED

FIELD    NAME        TYPE    LENGTH      DEC
01       NAME         C      030
02       ADDRESS      C      030
03       CITY         C      020
04       STATE        C      002
05       ZIP          C      005
06       AMOUNT       N      007          2

TOTAL BYTES:                 00094   ok

COPY STRUCTURE EXTENDED TO Temp
00004 RECORD(S) COPIED ok

USE Temp ok
DISPLAY STRU

STRUCTURE FOR TEMP.DAT
INDEX IN USE: NONE
```

Copy, cont.

```
NUMBER OF RECORDS: 4
PRIMARY SELECTED

FIELD       NAME          TYPE      LENGTH        DEC
01          NAME           C        010
02          TYPE           C        001
03          LENGTH         N        003
04          DEC            N        003

TOTAL BYTES:                        00018   ok

LIST
00001       NAME           C        030            0
00002       ADDRESS        C        030            0
00003       CITY           C        020            0
00004       STATE          C        002            0
00005       ZIP            C        005            0
00006       AMOUNT         N        007            2     ok
```

Count

FULL FORM: COUNT [{scope}] [FOR {exp}] [TO {memvar}] [WHILE {exp}]

PURPOSE: Used for counting records in a USE file

OVERVIEW: The COUNT command is used to determine how many records there are in a file (or part of a file). The results of a COUNT are displayed on the display in this form: "COUNT = xxxxx"

The COUNT command can also be used to determine how many records in a file satisfy a certain condition or set of conditions. The optional "FOR {exp}" clause is used for this purpose.

EXAMPLE #1: A FOR statement in association with a COUNT command:

```
USE Names
LIST ok

00001   HOWARD, ROY         2387 W. Benedict
     Miami      FL  27865  10235
00002   THOMAS, BRENDA   87 Laurel - Apt. #5
     Chicago    IL  48976  76926
00003   MURPHY, VINCE       987-B Hallowell
     Phoenix    AZ  87356  58938
00004   ALIMBO, JENNY       9376 Main
     Seattle    WA  99876  38769
00005   HAMILTON, JOHN      728 Downey Place
     Dallas     TX  49039  02678
00006   HORMAN, EILEEN      2 E. Plaskett Ct.
     Phoenix    AZ  87356  58397
00007   CUCUK, CHERYL       746 Manhassett Lane
     San Jose   CA  94687  10003
00008   ZACHRY, DAVE        P.O. Box 93876
     Reno       NV  93713  20835
00009   FAORO, ASHLEY       8467 Quinton
     Denver     CO  82076  69361
00010   MADDEN, KRIS        24 S. Broadway
     Boston     MA  02815  48376   ok

COUNT FOR Zip:Code > 50000
COUNT = 00006 ok
```

Count, cont.

Users have option of returning the results of the COUNT to a memory variable as well as the display. If the "TO {memvar}" clause is included in the COUNT statement, the integer count is placed into a memory variable. The memory variable will be created if it did not exist prior to this command.

EXAMPLE #2: The results of a COUNT are stored in a memory variable as well as displayed on the screen:

```
USE Names ok
COUNT TO MZip FOR ZIP>50000
COUNT = 00006 ok
? MZip
6 ok
```

The COUNT command will count deleted records if SET DELETED is "OFF," and ignore them if SET DELETED is "ON."

Create

FULL FORMS:

1.) CREATE [{filename}]
[{filename} FROM {filename}]

2.) CREATE FOLDER {folder name}

PURPOSE:

Used to establish a database file of the type ".DAT" and to create a "folder" on the default drive

OVERVIEW:

The first form of the CREATE command is used to establish the structure of a new **H & D Base** data file. After issuing it from the COMMAND LEVEL of the program, the user is prompted for the structure of the new file. The stucture includes the name of the file, and the name, type, length, and number of decimal places of each field to be included in it.

The prompts appear in the following order:

FILENAME:

1.) Must start with a letter (not a number or symbol)
2.) Can be up to 8 characters long
3.) Cannot include any colons and/or blank spaces

```
ENTER RECORD STRUCTURE AS FOLLOWS:
FIELD        NAME,TYPE,LENGTH,DECIMAL DIGITS
001
```

The information for each field must be entered as specified; e.g., each piece of information separated with a comma (and no spaces). It is not necessary to include a decimal place designation.

1.) NAME: A character string of up to 10 characters; alpha or numeric characters and colons *only*. Must begin with an alphabetic character.

Create, cont.

2.) TYPE: Three types possible -- character ("C"), numeric ("N"), or logical ("L")

3.) LENGTH: The length of the field -- any number from 1 to 254. Logical data may only be one character in length.

4.) DECIMAL DIGITS: Any number from 1 to 8, as long as the number is at least one digit less than the total width of the field. You need to specify the number of decimal digits only on fields which are numeric in nature.

The process is terminated when a {CR} is issued instead of the name of a field. The empty file is stored under the name specified upon entry with a ".DAT" trailer.

EXAMPLE #1: The creation of a simple database file:

```
CREATE

ENTER NAME OF NEW FILE: Names
ENTER THE FIELDS FOR THE NEW DATABASE:
NAME     - NAME OF FIELD, MAXIMUM LENGTH = 10
TYPE     - CHARACTER (C), NUMERIC (N), OR LOGICAL (L)
LENGTH   - MAXIMUM LENGTH = 254
DECIMAL  - NUMBER OF DIGITS TO LEFT OF DECIMAL POINT

FIELD NO.    NAME,TYPE,LENGTH,NO. OF DECIMAL DIGITS

01           Name,C,30
02           Address,C,20
03           City,C,15
04           State,C,2
05           Zip:Code,C,5
06           Acct:Num,N,5
07           {CR} ok

USE Names ok
DISPLAY STRUCTURE

STRUCTURE FOR NAMES.DAT
INDEX IN USE: NONE
NUMBER OF RECORDS: 0
PRIMARY SELECTED

FIELD        NAME          TYPE      LENGTH        DEC
01           NAME           C        030
02           ADDRESS        C        020
03           CITY           C        015
04           STATE          C        002
05           ZIP:CODE       C        005
06           ACCT:NUM       N        005
```

```
TOTAL BYTES:                              00078  ok
```

A database file so created must be USED before it can be manipulated in any manner.

The "CREATE {filename} FROM {filename}" command, is used to establish a new file by reading the structure of the FROM file. The contents of the FROM file must be input by the COPY STRUCTURE EXTENDED command.

EXAMPLE #2: A file CREATED from the structure of another file:

```
USE Names ok
COPY STRUCTURE EXTENDED TO Temp ok
CLEAR ok
CREATE Clients FROM Temp ok
```

The second form of the CREATE command is "CREATE FOLDER {folder name}". It is used to CREATE a "folder" on the the default drive. The command is very much like the "New Folder" command which can be executed from the Atari DESKTOP. Once a new "folder" has been created, data files of similar content can be stored in it (for example, all mailing list files or all payroll-related files). For further information see the "SET FOLDER" command.

EXAMPLE #3: The creation of a FOLDER:

```
CREATE FOLDER Payroll ok
```

Delete

FULL FORMS:

1.) DELETE [{scope}] [FOR{exp}]
[WHILE{exp}]

2.) DELETE FILE {filename}

PURPOSE: Used to mark a record for deletion or to eliminate a file

OVERVIEW: The first form of this command marks for eventual elimination all records which fall within the specified {scope}. Note: The default {scope} is the current record only.

Records marked for deletion will include an asterisk between the record number and first field when displayed.

If the optional "FOR" is used, only records which satisfy the {exp} will be marked for deletion.

EXAMPLE #1: The "FOR" clause in association with a DELETE:

```
USE NAMES ok
DELETE ALL FOR Acct:Num > 50000
00004 DELETION(S) ok
LIST

00001    HOWARD, ROY       2387 W. Benedict
     Miami       FL  27865  10235
00002   *THOMAS, BRENDA  87 Laurel - Apt. #5
     Chicago     IL  48976  76926
00003   *MURPHY, VINCE     987-B Hallowell
     Phoenix     AZ  87356  58938
00004    ALIMBO, JENNY     9376 Main
     Seattle     WA  99876  38769
00005    HAMILTON, JOHN    728 Downey Place
     Dallas      TX  49039  02678
00006   *HORMAN, EILEEN   2 E. Plaskett Ct.
     Phoenix     AZ  87356  58397
00007    CUCUK, CHERYL     746 Manhassett Lane
     San Jose    CA  94687  10003
00008    ZACHRY, DAVE       P.O. Box 93876
     Reno        NV  93713  20835
```

```
00009  *FAORO, ASHLEY     8467 Quinton
    Denver    CO  82076  69361
00010    MADDEN, KRIS      24 S. Broadway
    Boston    MA  02815  48376  ok
```

The RECALL operation can be used to "undelete" records marked for elimination with the DELETE command.

Records are not physically deleted until a PACK command is executed. Until that time, records marked for deletion can be displayed, but they cannot be COPIED or SORTED. Note: See the SET DELETED command for further information.

In the second form of the DELETE command, the designated {filename} will be removed from the disk drive where it resides, and the space it was occupying will be released to the operating system for reassignment. If, however, the {filename} is currently in use, the file will not be deleted and an error message will be generated. If a "trailer" is not specified, the program assumes that it is a file of the type ".DAT."

Once a file has been DELETED, there is no command available in **H & B Base** to retrieve it.

Display

FULL FORMS:

```
1.) DISPLAY [{scope}] [FOR {exp}] [{exp list}] [OFF]
                      [FIELDS {list}]
                      [WHILE <exp>]
2.) DISPLAY STRUCTURE
3.) DISPLAY MEMORY
4.) DISPLAY FILES [ON {disk drive}] [LIKE {skeleton}]
5.) DISPLAY STATUS
6.) DISPLAY COMMAND {command file}
```

PURPOSE: Used to show data, database structure, memory contents, files, system status, and the commands of a COMMAND FILE

OVERVIEW: The real purpose of any database management system is the DISPLAY of stored information in a wide variety of forms. This command is, therefore, one of the basic building blocks of **H & D Base**. The first form of the command allows you to DISPLAY all or part of the database in USE. If neither a {scope} nor a FOR {exp} is specified in the command, only the current record can contribute information for display. If {scope} is not specified and there is a FOR {exp}, then all records in the database may contribute to the display. If both the {scope} *and* FOR clauses are specified, then only those records that satisfy the FOR's conditional expression can contribute information for display.

As for fields, they are all displayed unless: 1) an {exp list} clause is used, or 2) you list the fields. Valid expressions may consist of data fields, memory variables, or any valid literal number, character or logical value.

The number of each record will appear before the information of the record displayed *unless* the optional "OFF" is included in the statement. DISPLAY FIELDS {list} allows you to use (and DISPLAY) field names that are otherwise ambiguous i.e., STATUS, FILE, STRUCTURE.

Assuming that there are more than 15 records which satisfy the conditions of the entire DISPLAY statement, the command, when executed, will DISPLAY the first 15 records then pause. Press any key to continue to the next 15 (to the end of the file). To return to the COMMAND LEVEL of the program during a display, press the "Escape" key {ESC}.

The LIST command is identical to the DISPLAY command except that LIST does not wait after each 15 records and its default {scope} is ALL records.

EXAMPLE #1: The DISPLAY of selected fields of a {scope} of records:

```
USE NAMES ok
DISPLAY NEXT 5 FIELDS Name,$(City,1,4),Acct:Num
00001  HOWARD, ROY          Miam   10235
00002  THOMAS, BRENDA       Chic   76926
00003  MURPHY, VINCE        Phoe   58938
00004  ALIMBO, JENNY        Seat   38769
00005  HAMILTON, JOHN       Dall   02678   ok
```

EXAMPLE #2: The DISPLAY of all records in a file which satisfy a list of {expressions}:

```
USE NAMES ok
DISPLAY ALL FOR (City="Phoenix" .AND. State="AZ") .OR. Zip:
Code>=93713 OFF

MURPHY, VINCE             987-B Hallowell
   Phoenix    AZ  87356   58938
ALIMBO, JENNY             9376 Main
   Seattle    WA  99876   38769
HORMAN, EILEEN            2 E. Plaskett Ct.
   Phoenix    AZ  87356   58397
CUCUK, CHERYL             746 Manhassett Lane
   San Jose   CA  94687   10003   ok
```

In the second form of the DISPLAY command, "DISPLAY STRUCTURE", only the *STRUCTURE* of the database in USE is displayed.

EXAMPLE #3: The DISPLAY of a file's structure:

```
USE Names ok
DISPLAY STRU

STRUCTURE FOR NAMES.DAT
INDEX IN USE: NONE
NUMBER OF RECORDS: 10
PRIMARY SELECTED
```

Display, cont.

```
FIELD      NAME          TYPE     LENGTH      DEC
01         NAME           C       030
02         ADDRESS        C       020
03         CITY           C       015
04         STATE          C       002
05         ZIP:CODE       C       005
06         ACCT:NUM       N       005

TOTAL BYTES:                      00078  ok
```

The third form of the DISPLAY command ("DISPLAY MEMORY") allows you to look at the state of all current memory variables. Their names and values are DISPLAYED.

EXAMPLE #4: The DISPLAY of all current memory variables:

```
STORE 'OLIVER' TO MName ok
STORE 2468 TO MOrder ok
STORE Y TO MLog ok
DISPLAY MEMORY

MNAME       (C)    OLIVER
MORDER      (N)    2468
MLOG        (L)    .T.
**TOTAL**          03 VARIABLES USED    23 BYTES USED    ok
```

The fourth form of the DISPLAY command, "DISPLAY FILES", is used to provide a listing of the ".DAT" files on either the default disk drive or an alternate {disk drive}. Relevent statistics on the files are also displayed.

The "LIKE {skeleton}" phrase allows other types of files to be displayed.

EXAMPLE #5: The DISPLAY of all ".DAT" files on drive "B":

```
DISPLAY FILES ON B LIKE *.DAT
```

EXAMPLE #6: The DISPLAY of all files on drive "C" beginning with the characters "ACCT":

```
DISPLAY FILES ON C LIKE ACCT*.*
```

DISPLAY STATUS, The fifth form of the DISPLAY command, allows you to show any files that are in USE (primary and secondary) along with any index files and "Key" expressions that are in use. DISPLAY STATUS also shows the current settings of all SET commands.

Using the "DISPLAY COMMAND {command file}" form, you can list the commands of the specified {command file} on the screen. The commands will also be sent to the printer if "SET PRINT" has been turned "ON". This is the handiest way to get a listing of the code which comprises a COMMAND FILE.

Do

FULL FORMS:

```
1.) DO {file}
2.) DO WHILE {exp}
       {statements}
    ENDDO
3.) DO CASE
      CASE {exp}
         {statements}
    CASE {exp}
         {statements}
      CASE {exp}
         {statements}
      [OTHERWISE]
         {statements}
    ENDCASE
```

PURPOSE: Used to either initiate the execution of a COMMAND FILE or to perform a group of commands over and over

OVERVIEW: When the first form of the DO command ("DO {file}") is either entered from the keyboard or encountered in a COMMAND FILE, the COMMAND {file} specified in the statement is opened and the commands of that {file} are executed.

The process continues until one of three situations occurs:

1.) The end of the file is reached.
2.) A RETURN command is encountered.
3.) A CANCEL command is encountered.

In situations #1 and #2, control is returned to either the COMMAND LEVEL of the program *or* the COMMAND FILE which called it. In the case of a CANCEL command being encountered, all COMMAND FILES are closed and control automatically returns to the COMMAND LEVEL.

COMMAND FILES can contain "DO" statements which intitate the execution of other COMMAND FILES. They can be "nested" up to 16 levels deep. In the "DO WHILE {exp}" command, if the {exp} evaluates as a logical "true", the statements following the DO are executed until an ENDDO statement is encountered. If the {exp} evaluates to a logical "false", control is transferred to the statement following the ENDDO statement. If there is an IF "inside" a DO WHILE, then an ENDDO may not occur before the ENDIF.

EXAMPLE #1: A simple "DO WHILE {exp}" command sequence:

```
DO WHILE .NOT. EOF
     DISPLAY Company
     SKIP
ENDDO
```

The third form of the "DO" command is "DO CASE". It is an extension of the DO command and takes the form shown above. It can best be described as a series of "IF...ENDIF" statements.

When a "DO CASE" is encountered in a COMMAND FILE, the program will examine the expression of the first CASE to determine if it is logically "true". If it is, the statements after it will be executed. When the next phrase beginning with "CASE" is reached, it will skip to the ENDCASE. If more than one CASE is true, only the first one will be executed.

If the first CASE is evaluated as logically "false" according to the {exp}, the program will move to the next CASE. If none of the CASES are "true", the DO CASE will be exited with *none* of the {statements} being executed. If the optional OTHERWISE clause is present and none of the CASEs are "true", the {statements} in the OTHERWISE clause will be executed. There is no limit to the number of CASE phrases that a DO CASE may contain, but you are limited to 16 "nested" DO CASES.

EXAMPLE #2: A DO CASE command comparing strings:

Command File:

```
STORE 'Mike' TO MName
DO CASE
```

Do, cont.

```
                CASE MName = "Tom"
                      STORE 1 TO MT
                CASE MName = "Sue"
                      STORE 2 TO MT
                CASE MName = "Mike"
                      STORE 3 TO MT
        ENDCASE
        DISPLAY MEMORY
```

Results:

```
MNAME         (C)    Mike
MT            (N)    3

**TOTAL**            02 VARIBLES USED        20 BYTES USED    ok
```

EXAMPLE #3: A DO CASE command comparing strings using OTHERWISE:

Command File:

```
STORE 'Mike' TO MName
DO CASE
        CASE MName = "Tom"
              STORE 1 TO MT
        CASE MName = "Sue"
              STORE 2 TO MT
        CASE MName = "Fred"
              STORE 3 TO MT
        OTHERWISE
              STORE 4 TO MT
ENDCASE
DISPLAY MEMORY
```

Results:

```
MNAME         (C)    Mike
MT            (N)    4
**TOTAL**            02 VARIBLES USED        20 BYTES USED    ok
```

Edit

FULL FORM: EDIT [n]

PURPOSE: Used to alter the data in a database

OVERVIEW: The EDIT command allows the user to alter the contents of the records in a database. If the command is entered without the optional [*n*] (for record #), the program will place the contents of the current record on the screen. If a record # *is* specified, that record will appear on the screen.

The following control and cursor movement keys can be used in the EDIT mode:

1.) {UP ARROW} or {CTRL-E}
Moves the cursor one field up
2.) {DOWN ARROW} or {CTRL-X}
Moves the cursor one field down
3.) {LEFT ARROW} or {CTRL-S}
Moves the cursor one character to the left
4.) {RIGHT ARROW} or {CTRL-D}
Moves the cursor one character to the right
5.) {HOME}
Moves the cursor to the first character in the record
6.) {CTRL-Y}
Deletes line
7.) {CTRL-T}
Deletes from cursor to end of line
8.) {CTRL-G} or {DELETE}
Delete character under cursor
9.) {BACKSPACE}
Delete character to left of cursor
10.) {CTRL-U}
Delete record toggle

Edit, cont.

11.) {CTRL-A} or {CTRL-R}
Save record - Back up to previous record

12.) {CTRL-F} or {CTRL-C}
Save record - Advance to next record

13.) {CTRL-W}
Exit - Save changes

14.) {CTRL-Q}
Exit - Do *not* save changes

Special Note: Atari has chosen to designate {CTRL-C} as the code for exiting from a program and returning to the DESKTOP. It is "hard wired," but can be overridden (to a certain degree) by software programmers. If you press {CTRL-C} two or more times *in rapid succession*, you will automatically exit **H & D Base** and return to the DESKTOP. We have included {CTRL-C} only as a convenience for those who use it extensively in other programs.

If a "SET FORMAT TO {file}" is in effect, EDIT will use the "@" commands from the format file to form the screen display and control the data that may be changed. Otherwise, EDIT displays all fields in tabular form.

If the file EDITED is being used with an associated index file (or files), the program will adjust the index *if the "Key" field is altered*. Any index file *not* selected when its "Key" is altered will *not* be updated and must be recreated.

Eject

FULL FORM: EJECT

PURPOSE: Used to issue a form feed on the printer

OVERVIEW: If "SET PRINT" is "ON", or the "FORMAT has been SET TO PRINT", this command will cause the printer to eject the page (form feed).

If the "@" command is being used to do direct page formatting, the EJECT command also zeros the line and column registers.

Refer also to the "SET EJECT ON/OFF" command.

Erase

FULL FORM: ERASE

PURPOSE: Used to clear the console screen

OVERVIEW: When this command is issued either from the COMMAND LEVEL or PROGRAM LEVEL of **H & D Base**, the screen is cleared and the cursor is placed in the upper left corner.

If it is issued when one or more "@" commands are "active" and the FORMAT is SET TO the SCREEN, all GETS and PICTURES will be cleared from memory.

Find

FULL FORM: FIND {char string} or {'char string'}

PURPOSE: Used to locate a record in an indexed database

OVERVIEW: The FIND command is the fastest way to search for information in a database. When it is issued, **H & D Base** will locate the *first* record in an (actively) *indexed* database with the same contents in the "Key" field as those specified in the {char string}. The database *must* be indexed, the appropriate index *must* be in USE, and the {char string} *must* appear in the "Key" string. (See the INDEX section for further information on "Keys").

EXAMPLE #1: Two applications of FIND:

```
USE Names ok
INDEX ON Zip:Code TO Zipind ok
USE Names INDEX Zipind ok
FIND 49039 ok
DISPLAY

00005   HAMILTON, JOHN       728 Downey Place
    Dallas      TX   49039   02678   ok

STORE 02815 TO MTemp ok
FIND &MTemp ok
DISPLAY

00010   MADDEN, KRIS         24 S. Broadway
    Boston      MA   02815   48376   ok
```

If the "Key" field is "Character" in nature, the FIND will operate even if it is given only the first few characters of the field. For example, "DUCK" may be entered when searching for "DUCKWORTH". If, however, another entry with the same four beginning letters appears in the file before "DUCKWORTH", that record will be placed on the screen (example: "DUCKINGTON").

EXAMPLE #2: A FIND using only the first few characters of a field:

```
USE Names ok
```

Find, cont.

```
INDEX ON Name to Nameind ok
SET INDEX TO Nameind ok
FIND ZACH ok
DISPLAY NAME OFF

ZACHRY, DAVE                P.O. Box 93876
    Reno        NV   93713   20835   ok
```

If the index is "Keyed" to a numeric field, the found record will be the first record whose key is *arithmetically equal* to the object of the FIND.

Note: For indexes keyed on both characters and numbers, the FIND object is a character string with or without quotation marks. Quotation marks only become necessary for character strings *if the original key had leading blanks*. In that case, the exact number of leading blanks should be inside the quotes.

It is possible to use a memory variable as the object of a FIND. To do so, use "Macro Substitution." The {memvar} must be placed after the FIND command using the ampersand symbol (&) in this form: FIND &{memvar}. For more information, see "&".

Once a record in a database has been located using the FIND command, it can be processed just as any other database record using a variety of **H & D Base** commands. Commands which cause movement of the database (like DISPLAY, COPY, etc.) will process the found record first and proceed to the next record in sequence, based upon the key.

If no record exists whose key is identical to the {char string} the message: "NO FIND" will be displayed on the screen and the record number function "#" will return the value of zero.

If the found record is *not* the one which was desired, but one which appears before it, use the SKIP or "LOCATE FOR {exp}" commands to see if any more matches exist.

If SET EXACT is "ON", FINDS will only be made if there is a *character for character* match for the ENTIRE key (except for trailing blanks).

IF SET DELETE is "ON" the FIND will *ignore* any record which has been marked for deletion by the DELETE command.

Functions

FULL FORM: Does not apply

PURPOSE: Used to perform special purpose operations in expressions

OVERVIEW: Some of the operations necessary for the effective manipulation of data can only be performed through the use of the special purpose FUNCTIONS available to you in **H & D Base**. They can be run from either the COMMAND LEVEL or the PROGRAMMING LEVEL of **H & D Base**. The parenthesis ( ) in the formulas *must* be used.

This section provides a brief summary of each of those functions. They are listed alphabetically.

Blank String Function

SPACE(*nnn*)

This function returns a character string of *nnn* spaces.

```
STORE SPACE(10) TO MStr
```

The memory variable "MStr" will now be a 10 character string of blanks.

Date Function

DATE( )

This FUNCTION will generate a character string that contains the date stored in the Atari computer (as established in the DESKTOP) in the format XX/XX/XX. The characters should be entered exactly as shown (without anything between the parenthesis). The character string always has a length of 8 characters.

```
? DATE( )
12/15/85 ok
SET DATE TO 01/01/86 ok
```

```
? DATE( )
01/01/86 ok
```

The date stored in the Atari system can be changed from the Atari DESKTOP or by using the SET DATE TO command.

Decimal Place Function

DEC({numeric expression},{decimal places})

This function sets the decimal position of the {numeric expression} to the number specified in {decimal places}.

```
STORE 35.768 TO MNum ok
? DEC(MNum,2)
35.76 ok
```

Deleted Record Function

*****

This FUNCTION delivers a logical "True" (.T.) if the current record has been marked for deletion, and a logical "False" (.F.) if has not.

```
USE People ok
DELETE RECORD 1
00001 DELETION(S) ok
? *
.T. ok
SKIP ok
DISPLAY #
00002 ok
? *
.F. ok
```

End-of-File Function

EOF

This FUNCTION is used to determine if the end of the file has been reached. It delivers a logical "True" (.T.) if it has, and a logical "False" (.F.) if it has not.

```
USE People ok
? EOF
.F. ok
GOTO BOTT ok
SKIP ok
? EOF
.T. ok
```

File Function

FILE({"filename"/variable/expression})

This FUNCTION will tell you if a certain file exists on the disk. It will generate a logical "True" (.T.) if it is, and a logical "False" (.N.) if it isn't.

```
DISPLAY FILES LIKE *.*

                    # RECS        BYTES    DATE       TIME
ADDACCT.CMD      Not a data file   2314   11-28-85   06:48 pm
NAMES.DAT            00010         2818   01-05-86   12:45 pm
NAMEIND.NDX      Not a data file   6444   01-06-86   01:15 pm
LEDGER.DAT           00015         2453   12-24-85   11:24 pm    ok

? FILE("Names.DAT")
.T. ok

? FILE("People.DAT")
.F. ok
```

Integer Function

INT({numeric expression})

This FUNCTION takes a number with decimals and *eliminates* everything to the right side of the decimal point.

To "round off" a number with a decimal to the nearest whole number, use this form of the INT FUNCTION:

```
INT(value + .5)
```

```
? INT(123456.789)
123456 ok
STORE 123456.789 TO MNum ok
? INT(MNum)
123456 ok
? INT(123456.789 + .5)
123457 ok
```

Integer to String Function

STR({expression/variable/number}, {length},{decimals})

This FUNCTION converts a number (or contents of a numeric variable) into a string with a specified length and a specified number of digits to the right of the decimal point. The length you specify *must* be large enough to hold at least all the digits plus the decimal point.

Functions, cont.

```
? STR(12,3,1)
1.2 ok
? STR(123456789,10,5)
1234.56789 ok
STORE 123.50 TO MT
ok
? "$" + STR(MT,6,2)
$123.50 ok
```

Number to Character Function

CHR({number})

This FUNCTION yields the ASCII *character equivalent* of a specified number.

```
? CHR(65)
A ok
? CHR(36)
$ ok
```

Rank Function

RANK({string})

This FUNCTION is used to return the ASCII value of the *first* character of a string.

```
? RANK("A")
65 ok
? RANK("$BM%K@B")
36 ok
```

Record Function

#

This FUNCTION delivers the record number of the current file.

```
USE People ok
DISPLAY Name

00001 CUCUK, CHERYL ok
? #
1 ok
GOTO BOTT ok
DISPLAY #
10 ok
?? # 10 ok
```

String Length Function

LEN(variable/string})

This FUNCTION tells you how many characters there are in the string you name. Note: If a character field variable name is used, it will tell you how many characters are in the *entire* field.

```
STORE "abcdefghijklmnopqrstuvwxyz" TO MAlpha ok
? LEN(MAlpha)
26 ok

USE Names ok
GO 7 ok
DISPLAY

00007   CUCUK, CHERYL      746 Manhassett Lane
     San Jose   CA   94687   10003   ok
```

String to Integer Function

VAL({char string})

This FUNCTION converts a character string (or substring) into a number of equal quantity. The string can be made up of digits, a sign, and up to one decimal point.

If the character string begins with numeric characters but also contains non-numeric characters, the value generated by the VAL function will equal the leading numeric characters.

```
? VAL("123")
123 ok

? VAL("ABC")
0 ok

? VAL("123ABC")
123 ok

? VAL("123ABC") + VAL("456DEF")
579 ok
```

The ampersand symbol (&) can also be used to convert strings to numeric values (only in COMMAND FILES).

Command File:

```
STORE "123.25" TO MNum
? 10.25 + &MNum
```

Results:

```
133.50
```

Functions, cont.

Substring Function

$({expression/variable/string},{start},{length})

This FUNCTION forms a character string which is comprised of all or part of another string. The new string begins from the character in the position specified by {start}, and continues for {length} number of characters. {start} and {length} must be literals, variables, or expressions.

```
? $("abcdefghijklmnopqrstuvwxyz",1,5)
abcde ok
? $("abcdefghijklmnopqrstuvwxyz",6,10)
fghijklmno ok

STORE 6 TO M1 ok
STORE 10 TO M2 ok
? $("abcdefghijklmnopqrstuvwxyz",M1,M2)
fghijklmno ok

USE Names ok
LOCATE FOR $(Name,9,5) = "JENNY"
RECORD 00004 ok
DISPLAY
00004   ALIMBO, JENNY      9376 Main
     Seattle     WA  99876  38769  ok
```

When this function is used to generate a "Key" for indexing, the specifiers *must* be literals.

Substring Search Function

@({variable 1/string 1},{variable 2/string 2})

This FUNCTION yields an integer which represents the beginning position of one string within another string. If the first string does not appear in the second, a value of "0" will be issued.

```
? @("m","abcdefghijklmnopqrstuvwxyz")
13 ok
? @("xyz","abcdefghijklmnopqrstuvwxyz")
24 ok
```

Trim Function

TRIM({string})

This FUNCTION is used to eliminate all trailing blanks in the contents of a string variable. It should be used in this form:

```
STORE TRIM({variable}) TO {variable}
STORE "MISSISSIPPI                       " TO MState ok
? LEN(MState)
34 ok
STORE TRIM(MState) TO MState ok
? LEN(MState)
11 ok
```

Note: This function must *not* be used in the INDEX command.

Type Function

TYPE({expression})

This FUNCTION yields a single character string that contains a "C", "N", "L", or "U" depending on whether the data in the expression is "Character", "Numeric", "Logical", or "Undefined" (respectively).

```
STORE 123456789 TO MNumber ok
STORE "ABCEDFGHI" TO MString ok
STORE Y TO MLogic ok
? TYPE(MNumber)
N ok
? TYPE(MString)
C ok
? TYPE(MLogic)
L ok
```

Uppercase Function

!({variable/string})

This FUNCTION changes all the *lower case* alphabetic characters in a string or string variable into *upper case* characters.

```
? !("aBcDeFgHiJkLmNoPqRsTuVwXyZ")
ABCDEFGHIJKLMNOPQRSTUVWXYZ ok
```

GO or GOTO

FULL FORMS:

1.) GOTO [RECORD] {*n*}

2.) GOTO TOP

3.) GOTO BOTTOM

PURPOSE: Used to re-position the record pointer

OVERVIEW: When executed, this first form of this command postions the pointer to record {*n*}. The word "GO" can be substituted for "GOTO" at will, and the word "RECORD" is completely optional.

EXAMPLE #1: Re-positioning the pointer with the "GOTO {*n*}" command:

```
USE Names ok
DISPLAY

00001   HAMILTON, JOHN      2387 W. Benedict
    Miami       FL  27865   10235   ok

GOTO 5 ok
DISPLAY

00005   HAMILTON, JOHN      728 Downey Place
    Dallas      TX  49039   02678   ok

GO RECORD 10 ok
DISPLAY

00010   MADDEN, KRIS        24 S. Broadway
    Boston      MA  02815   48376   ok
```

In the second form of the "GOTO" command ("GOTO TOP") places the pointer on the first record in the file (TOP). "GOTO BOTTOM" places it on the last record in the file (BOTTOM). Important note: When the file has been indexed, these are first/last records *according to the "Key" used to index the database*.

Help

FULL FORM: HELP {command}

PURPOSE: Used in situations where a short explanation of a particular **H & D Base** command is needed.

OVERVIEW: The proper syntax for any **H & D Base** command (along with a short explanation of its execution) can be obtained by entering "HELP {command}" from the COMMAND LEVEL of the program. {command} can only be the *first* whole word of any **H & D Base** statement.

When executed, the program will search the default drive for a file named "HDBASE.HLP". When it is found, the information it includes regarding the specified {command} will be placed on the screen. In some instances, the information regarding a particular command will not fit completely on one screen. If this is the case, press any key to move to the next screen. If the {command} is *not* included in the "HELP" file, or the "HELP" file does not reside on the disk in the default drive, an error prompt will appear on the screen.

Note: The "HDBASE.HLP" file is included on your Program Disk. We suggest that you transfer it to each of your Data Disks when they are first formatted (if you intend to use it regularly).

If

FULL FORM:

```
IF {exp}
       {commands}
[ELSE
       {commands}]
ENDIF
```

PURPOSE: Used for the conditional execution of a command or set of commands

OVERVIEW: When an IF command is encountered in a COMMAND FILE, the {exp} is evaluated as either "true" or "false". If evaluated "true", the commands following the IF are executed. If the {exp} is evaluated "false", the program skips to the ENDIF and continues from there.

If the optional ELSE statement is included and the {exp} evaluates as "true", the commands following the ELSE are skipped. If "false", the commands following the ELSE are executed.

IF commands may be nested to any level. Note: Statements must nest properly. For example, an IF with a nested DO WHILE must not end (with an ENDIF) *before* the DO WHILE ends.

EXAMPLE #1: A sample IF statement using the optional ELSE clause:

```
USE Ledger
DO WHILE .NOT. EOF
  IF BALANCE>0
    {any statement}
  ELSE
    {any statement}
  ENDIF
  SKIP
ENDDO
```

Index

FULL FORMS: 1.) INDEX ON {expression} TO {index file name} {ASCENDING} {DESCENDING}

2.) INDEX

PURPOSE: Used to create an index file for a database

OVERVIEW: The first form of the INDEX command is used to create an index file of the type ".NDX" which contains pointers to the records in the USE file. When used in association with the USE file, the records of the USE file appear to be in sorted order. The USE file, however, is not physically changed.

The {expression} used is called the "Key".

Sorting may be done in either ascending (the default) or descending order.

An INDEX file must be selected for use with the database from which it was created. The form:

USE {database filename} INDEX {index filename [,index filename, index...]}

EXAMPLE #1: A simple ascending INDEX on a character field:

```
USE Names ok
INDEX ON Name TO Nameidx
10 RECORDS INDEXED ok
USE Names INDEX Nameidx ok
LIST

00004   ALIMBO, JENNY       9376 Main
    Seattle     WA  99876  38769
00007   CUCUK, CHERYL       746 Manhassett Lane
    San Jose    CA  94687  10003
00009   FAORO, ASHLEY       8467 Quinton
    Boston      MA  02846  69361
```

Index, cont.

```
00005  HAMILTON, JOHN     728 Downey Place
   Dallas     TX  49039  02678
00006  HORMAN, EILEEN     2 E. Plaskett Ct.
   Phoenix    AZ  87356  58397
00001  HOWARD, ROY        2387 W. Benedict
   Miami      FL  27865  10235
00010  MADDEN, KRIS       24 S. Broadway
   Boston     MA  02815  48376
00003  MURPHY, VINCE      987-B Hallowell
   Phoenix    AZ  87356  58938
00002  THOMAS, BRENDA     87 Laurel - Apt. #5
   Chicago    IL  48976  76926
00008  ZACHRY, DAVE       P.O. Box 93876
   Reno       NV  93713  20835  ok
```

You may INDEX on more than one field by using the command in this form:

```
INDEX ON {field #1} + {field #2} + {field #3} TO {file}
```

{field #1} receives the highest priority -- {field #3} the lowest.

EXAMPLE #2: An INDEX on a more involved {expression}:

```
USE Names ok
INDEX ON City + Name TO Cityidx
10 RECORDS INDEXED ok
SET INDEX TO Cityidx ok
LIST

00009  FAORO, ASHLEY      8467 Quinton
   Boston     MA  02846  69361
00010  MADDEN, KRIS       24 S. Broadway
   Boston     MA  02815  48376
00002  THOMAS, BRENDA     87 Laurel - Apt. #5
   Chicago    IL  48976  76926
00005  HAMILTON, JOHN     728 Downey Place
   Dallas     TX  49039  02678
00001  HOWARD, ROY        2387 W. Benedict
   Miami      FL  27865  10235
00006  HORMAN, EILEEN     2 E. Plaskett Ct.
   Phoenix    AZ  87356  58397
00003  MURPHY, VINCE      987-B Hallowell
   Phoenix    AZ  87356  58938
00008  ZACHRY, DAVE       P.O. Box 93876
   Reno       NV  93713  20835
00007  CUCUK, CHERYL      746 Manhassett Lane
   San Jose   CA  94687  10003
00004  ALIMBO, JENNY      9376 Main
   Seattle    WA  99876  38769  ok
```

Any number of INDEX files may be created for a database. Any INDEX which is in USE during an APPEND, EDIT, REPLACE, READ, or BROWSE will *automatically* be updated if a change has been made to the "Key". Any INDEX *not* in USE will not be updated and must, therefore, be RE-INDEXED.

INDEXING allows very rapid location of database records by specifying all or part of the "Key" by means of the FIND command. (See FIND).

Refer also to the "REINDEX" and "SET INDEX" commands.

Two notes of caution:

1.) Do not use the TRIM function as part of an INDEX "Key". This will create INDEX "Keys" of variable length which is illegal. All INDEX "Keys" must be equal in length.

2.) If the "$" or "STR" functions are used as part or all of a "Key", they must have literal numbers (not variables or expressions) as their length parameters.

The second form of the INDEX command ("INDEX") INDEXES the current record to the INDEX files in use. This allows an INDEX to be built which does not include all the records in a database.

Input

FULL FORM: INPUT ["{cstring}"] TO {memvar}

PURPOSE: Used to place user input into memory variables

OVERVIEW: The INPUT command can be issued from either the COMMAND or PROGRAM LEVELS of **H & D Base**. It is used to facilitate the entry of expression values into memory variables.

If the optional {cstring} is used, the characters are displayed on the screen as a prompt message before the input is accepted. User response (up to 254 characters) is placed in {memvar}. If the {memvar} does not exist when the INPUT command is encountered, it is created.

The "type" of the {memvar} is determined from the type of data that is entered. Character strings must be delimited with either single or double quotes. A response of this type results in a {memvar} of "Character" type. If a numeric expression is entered, the {memvar} will be "Numeric" type. If a "T" or "Y" (for "True" or "Yes") is entered, {memvar} will be a "Logical" variable with the value TRUE; if an "F" or "N" (for "False" or "No") is entered, {memvar} will be a "Logical" variable with the value "FALSE". Note: the TYPE function may be used to determine the type of the entry.

INPUT should be used to enter numeric and logical data only. The ACCEPT command is a more convenient way to enter character strings.

EXAMPLE #1: INPUTS with prompts:

```
INPUT "AMOUNT OF SALE" TO MSale 285.46 ok
?? MSale 285.46 ok
INPUT "ENTER T OR F" TO MRes T ok
?? MRes .T. ok
```

Insert

FULL FORMS: 1.) INSERT [BEFORE]

2.) INSERT BLANK [BEFORE]

PURPOSE: Used to add a record among other records in a database

OVERVIEW: The first form of the INSERT command is a specialized form of the APPEND command which is used to place a *single* record at a position in a database *other than at the bottom*. Its primary use is to keep a file which has been SORTED in the proper order.

When issued, the user will be prompted for input values in the same manner as the APPEND mode (the same cursor and control functions apply). When the process has been completed, the record is INSERTED into the file at a position *following* the current record and all following record numbers are adjusted. If the {BEFORE} option is specified, the record is INSERTED at a position *before* the current record and all record numbers which follow are adjusted.

EXAMPLE #1: A record INSERTED into a database file after the current record.

```
USE Names ok
LIST

00001   HOWARD, ROY        2387 W. Benedict
    Miami      FL  27865  10235
00002   THOMAS, BRENDA   87 Laurel - Apt. #5
    Chicago    IL  48976  76926
00003   MURPHY, VINCE      987-B Hallowell
    Phoenix    AZ  87356  58938
00004   ALIMBO, JENNY      9376 Main
    Seattle    WA  99876  38769
00005   HAMILTON, JOHN     728 Downey Place
    Dallas     TX  49039  02678
00006   HORMAN, EILEEN    2 E. Plaskett Ct.
    Phoenix    AZ  87356  58397
00007   CUCUK, CHERYL     746 Manhassett Lane
    San Jose   CA  94687  10003
```

Insert, cont.

```
00008   ZACHRY, DAVE          P.O. Box 93876
    Reno        NV  93713   20835
00009   FAORO, ASHLEY         8467 Quinton
    Denver      CO  82076   69361
00010   MADDEN, KRIS          24 S. Broadway
    Boston      MA  02815   48376   ok

GOTO RECORD 7 ok
INSERT

RECORD 00008

NAME          :FRUMP, YAGO              :
ADDRESS       :1987 Woodley             :
CITY          :Houston        :
STATE         :TX:
ZIP:CODE      :42103:
ACCT:NUM      :58367: ok

LIST
00001   HOWARD, ROY           2387 W. Benedict
    Miami       FL  27865   10235
00002   THOMAS, BRENDA        87 Laurel - Apt. #5
    Chicago     IL  48976   76926
00003   MURPHY, VINCE         987-B Hallowell
    Phoenix     AZ  87356   58938
00004   ALIMBO, JENNY         9376 Main
    Seattle     WA  99876   38769
00005   HAMILTON, JOHN        728 Downey Place
    Dallas      TX  49039   02678
00006   HORMAN, EILEEN        2 E. Plaskett Ct.
    Phoenix     AZ  87356   58397
00007   CUCUK, CHERYL         746 Manhassett Lane
    San Jose    CA  94687   10003
00008   FRUMP, YAGO           1987 Woodley
    Houston     TX  42103   58367
00009   ZACHRY, DAVE          P.O. Box 93876
    Reno        NV  93713   20835
00010   FAORO, ASHLEY         8467 Quinton
    Denver      CO  82076   69361
00011   MADDEN, KRIS          24 S. Broadway
    Boston      MA  02815   48376   ok
```

As in the APPEND mode, if "SET CARRY" is SET "ON" when the INSERT command is instituted, the information in the previous record is carried over to the new record.

If "SET FORMAT TO {file}" is in effect, INSERT will use the "@" commands from the format file to form screen display and allow control of the screen and the data that will be appended. Otherwise, INSERT displays all fields in tabular form.

Notes on the "INSERT {BEFORE}" command:

1.) Any INSERT which is conducted on a large, non-indexed data base should be avoided as it must rewrite most of the file which takes a great deal of time.

2.) Any INSERT which is conducted on an indexed file will produce the same results as if it were being APPENDED.

The second form of the INSERT command is "INSERT BLANK [BEFORE]". It works identical to the first form except that it *automatically* INSERTS a *blank* record into the designated position instead of allowing you to enter information.

Join

FULL FORM: JOIN TO {file} FOR {expression} [FIELDS {field list}]

PURPOSE: Used to create a database composed of matching records from two other databases

OVERVIEW: The JOIN command can be used to create a third database using the records from two existing databases (PRIMARY and SECONDARY) whenever the terms of a certain {expression} are met.

Once the PRIMARY and SECONDARY databases have been determined, and the PRIMARY database is in USE, the JOIN command may be issued. It positions the program to the first record of the PRIMARY USE file and evaluates the "FOR {expression}" against *each* record in the SECONDARY USE file. Each time the {expression} yields a "True" result, a record is added to the new database. The new record is comprised of *all* fields of *both* the PRIMARY and SECONDARY databases *unless* the optional FIELDS clause is included (limit = 97 total). In that case, only specified FIELDS are included.

When all records of the SECONDARY file have been evaluated against the *first* record of the PRIMARY file, the PRIMARY file is advanced one record, and *all* records of the SECONDARY file are evaluated against the {expression} once again. The process continues until every record of the PRIMARY file has been evaluated against every record of the SECONDARY file.

IMPORTANT NOTES:

1.) The JOIN command takes a great deal of time to execute, especially when the databases involved include a large number of records.

2.) There is a strong possibility that the new database will be of unmanageable proportions if the specified {expression} allows an inordinate number of matches to be made. Do not make it too "loose."

3.) We strongly suggest that the name of a field used in *any* part of a JOIN statement include as a prefix to its name either a "P." (for a PRIMARY field) or an "S." (for a SECONDARY field). This will prevent any confusion on the program's part regarding the location of the field you are specifying.

EXAMPLE #1: A sample JOIN command:

```
USE Names ok
INDEX ON Name TO Nameind ok
SET INDEX TO Nameind ok
DISPLAY STRU

STRUCTURE FOR NAMES.DAT
INDEX IN USE: NAMEIND.NDX
KEY: NAME
NUMBER OF RECORDS: 10
PRIMARY SELECTED

FIELD     NAME          TYPE     LENGTH   DEC
01        NAME           C       030
02        ADDRESS        C       020
03        CITY           C       015
04        STATE          C       002
05        ZIP:CODE       C       005
06        ACCT:NUM       N       005

TOTAL BYTES:                     00078     ok

SELECT SECONDARY ok
USE Ledger ok

DISPLAY STRUCTURE

STRUCTURE FOR LEDGER.DAT
INDEX IN USE: NONE
NUMBER OF RECORDS: 15
SECONDARY SELECTED

FIELD     NAME          TYPE     LENGTH   DEC
01        ACCT:NUM       N       009
02        CUR:CHGS       N       009       2
03        BALANCE        N       009       2

TOTAL BYTES:                     00028     ok

SELECT PRIMARY ok
JOIN TO Temp FOR P.Acct:Num = S.Acct:Num FIELDS
P.Name,S.Cur:Chgs, S.Balance ok
```

Join, cont.

```
USE Temp ok
DISPLAY STRUCTURE

STRUCTURE FOR TEMP.DAT
INDEX IN USE: NONE
NUMBER OF RECORDS: 9
PRIMARY SELECTED

FIELD       NAME           TYPE      LENGTH   DEC
01          NAME             C       030
02          CUR:CHGS         N       009       2
03          BALANCE          N       009       2

TOTAL BYTES:                         00049     ok

LIST

00001      ALIMBO, JENNY     1083.36       8327.73
00002      CUCUK, CHERYL      267.92      10826.62
00003      FAORO, ASHLEY      356.01       9367.62
00004      HAMILTON, JOHN     105.83       2039.46
00005      HOWARD, ROY          4.29         83.83
00006      MADDEN, KRIS         9.37        728.98
00007      MURPHY, VINCE       52.27       2898.86
00008      THOMAS, BRENDA    8291.92     102930.83
00009      ZACHRY, DAVE        26.00        372.50   ok
```

List

FULL FORMS:

1.) LIST [{scope}] [FOR {exp}] [{exp list}] [OFF] [FIELDS {list}] [WHILE <exp>]

2.) LIST STRUCTURE

3.) LIST MEMORY

4.) LIST FILES [ON {disk drive}] [LIKE {skeleton}]

5.) LIST STATUS

6.) LIST COMMAND {command file}

PURPOSE:

Used to show data, database structure, memory contents, files, system status, and the commands of a COMMAND FILE.

OVERVIEW:

The first form of the LIST command is the same as the DISPLAY command with two notable exceptions:

1.) The {scope} defaults to ALL records
2.) LISTINGS are continuous (no pause after 15 records)

If "SET DELETED" is "ON", LIST will show only non-deleted records.

All other forms of the LIST command, including LIST MEMORY, LIST FILES, LIST STRUCTURE, and LIST COMMAND {command file}, work work exactly the same as the DISPLAY command.

Locate

FULL FORM: LOCATE [{scope}] [FOR {exp}]

PURPOSE: Used for finding a record in a database which matches a certain condition

OVERVIEW: When the LOCATE command is issued, the program searches the USE database sequentially for the first record whose data satisfies (e.g., delivers a logical "True") to the specified {expression}. If one is found, the number of that record is displayed on the screen (if TALK is SET "ON").

The CONTINUE command may be used to reinstitute the search. You may, however, execute other **H & D Base** commands between the LOCATE and the CONTINUE. CONTINUE ignores the {scope} issued with the LOCATE and continues to the end of the file.

EXAMPLE #1: Reinstituting a LOCATE with the CONTINUE command:

```
USE Names ok
LOCATE FOR Acct:Num > 50000
RECORD: 2 OK
DISPLAY

00002   THOMAS, BRENDA   87 Laurel - Apt. #5
    Chicago   IL   48976   76926   ok

CONTINUE
RECORD: 3 ok
DISPLAY

00003   MURPHY, VINCE       987-B Hallowell
    Phoenix   AZ   87356   58938   ok

CONTINUE
RECORD: 6 ok
CONTINUE
RECORD: 9 ok
CONTINUE
END OF LOCATE ok
```

If no record matching the {expression} is found, an "END OF LOCATE" message will be displayed, and the record pointer will be left in a position opposite the last record in the file. If the "NEXT *n*" clause (an option of {scope}) is specified, and no record which matches the {expression} can be found within the {scope} of the NEXT, the message "END OF LOCATE" will be displayed. The pointer will be positioned opposite the last record scanned.

Note: The LOCATE command works faster on a file that is USED without an INDEX file.

Loop

FULL FORM: LOOP

PURPOSE: Used as an escape mechanism for DO WHILE groups.

OVERVIEW: The LOOP command can only be used in a **H & D Base** COMMAND FILE as part of a DO WHILE group. It is used to shorten the DO WHILE loop which, if large, can be time consuming when run or may contain commands which are to be skipped at times.

When a LOOP command is encountered in the body of a DO WHILE, the remainder of the commands in the DO WHILE are skipped and program flow is returned directly to the DO WHILE statement. The commands comprising the body of the DO WHILE are then executed once again (if the {exp} is still "True").

EXAMPLE #1: A DO WHILE statement with an embedded LOOP:

Command File:

```
USE Names
STORE 'T' TO MCont
DO WHILE .NOT. EOF .AND. MCont = 'T'
        IF Acct:Num > 50000
            STORE 'F' TO MCont
            LOOP
        ELSE
            DISPLAY
            SKIP
        ENDIF
ENDDO
```

Something in the body of the DO WHILE *must* eventually change to deter flow of the program from the LOOP command or the looping process will continue forever.

Modify

FULL FORMS: 1.) MODIFY STRUCTURE
2.) MODIFY COMMAND [{file}]

PURPOSE: Used to alter the structure of a database or to enter/edit the instructions of a COMMAND FILE

OVERVIEW: The MODIFY STRUCTURE command is used to change the structure of a **H & D Base** data file (".DAT"). Any type of change may be made, including the addition and deletion of fields and the alteration of their parameters (NAME, TYPE, LENGTH, NUMBER OF DECIMAL DIGITS). Important Note: Any data resident in the USE file when the MODIFY STRUCTURE command is executed will be lost.

The database file MODIFIED with this command is the one in USE when it is issued from the COMMAND LEVEL of the program. When instituted, the existing structure of the database is displayed on the screen. Changes may then be made at will.

These are the cursor and control keys which will work with the MODIFY STRUCTURE command:

1.) {UP ARROW} or {CTRL-R} or {CTRL-A}
Moves the cursor one field up
2.) {DOWN ARROW} or {CTRL-C} or {CTRL-F}
Moves the cursor one field down
3.) {LEFT ARROW} or {CTRL-S}
Moves the cursor one character to the left
4.) {RIGHT ARROW} or {CTRL-D}
Moves the cursor one character to the right
5.) {CTRL-E} Moves the cursor one
section of field to left
6.) {CTRL-X}
Moves the cursor one *section* of field to right

Modify, cont.

7.) {CTRL-G} or {DELETE}
Delete character under cursor
8.) {BACKSPACE}
Delete character to left of cursor
9.) {CTRL-Y} or {CTRL-U}
Deletes field and moves all lower fields up
10.) {CTRL-N}
Moves all lines down to make room for new field
11.) {CTRL-W}
Exit - Save changes
12.) {CTRL-Q}
Exit - Do *not* save changes

Special Note: Atari has chosen to designate {CTRL-C} as the code for exiting from a program and returning to the DESKTOP. It is "hard wired," but can be overridden (to a certain degree) by software programmers. If you press {CTRL-C} two or more times *in rapid succession*, you will automatically exit **H & D Base** and return to the DESKTOP. We have included {CTRL-C} only as a convenience for those who use it extensively in other programs.

As noted previously, the MODIFY STRUCTURE command deletes *all* of the data resident in the USE file when it is executed. To modify the structure of a file with data, without harming that data, COPY the structure to a work file, USE the work file, make the modifications, and APPEND the old data to the work file. To conclude, DELETE the original file and RENAME the work file.

EXAMPLE #1: The process for MODIFYING a STRUCTURE without losing the data:

```
USE People ok
COPY STRUCTURE TO Temp ok
USE Temp ok
MODIFY STRUCTURE ok
APPEND FROM People ok
DELETE FILE People ok
RENAME Temp TO People ok
```

All records marked for DELETION will *not* be copied.

The second form of the MODIFY command, "MODIFY COMMAND [{file}], is used for editing COMMAND FILES. It can be issued only from the COMMAND LEVEL of **H & D Base**. If the file you specify does not exist, **H & D Base** will create it for you at this point.

The editing of a COMMAND FILE is facilitated through the "Command File Editor", a very simplistic word processor whose mode is entered automatically when the MODIFY COMMAND statement is registered. These are the control keys which will work with the "Text Editor":

1.) {UP ARROW} or {CTRL-E}
Moves the cursor one line up
2.) {DOWN ARROW} or {CTRL-X}
Moves the cursor one line down
3.) {LEFT ARROW} or {CTRL-S}
Moves the cursor one character to the left
4.) {RIGHT ARROW} or {CTRL-D}
Moves the cursor one character to the right
5.) {HOME} or {CTRL-O}
Moves the cursor to the first character in the Command File
6.) {CTRL-K}
Moves the cursor to the last character in the Command File
7.) {CR}
Inserts a line
8.) {CTRL-Y}
Deletes a line and moves all lines up
9.) {CTRL-G} or {DELETE} or {RIGHT BUTTON ON MOUSE}
Delete character under cursor
10.) {BACKSPACE} or {LEFT BUTTON ON MOUSE}
Delete character to left of cursor
11.) {UNDELETE}
Undelete last *backspaced* character
12.) {CTRL-W}
Exit - Save changes
13.) {CTRL-Q}
Exit - Do *not* save changes

Modify, cont.

In addition, you may use the "mouse" for positioning the cursor.

Up to 77 characters per line can be entered. If a statement consists of less than 77 characters, a {CR} will enter the statement, and move the cursor to the next line. If an attempt is made to enter a statement with more than 77 characters, part of the statement will automatically be "wrapped-around" to the next line (whole words only). A statement which does not fit entirely on one line must have a "tilde" (~) somewhere between the last character in the line and the right margin. Note: This is true only with COMMAND FILES. At the COMMAND LEVEL of the program, "tildes" at the end of continuous lines are not necessary. Just keep typing and the command will automatically wrap around to the next line.

To exit the "Command File Editor" and save any changes made in the program it contains, press {CTRL-W}. The file will be stored on the disk with a ".CMD" trailer (unless you have specified an alternate trailer). Use {CTRL-Q} to exit if changes are *not* to be saved.

Note or *

FULL FORMS: 1.) NOTE [{any characters}]

2.) * [{any characters}]

PURPOSE: Used to make notations in the body of a COMMAND FILE which will not be displayed when the file is "run"

OVERVIEW: The NOTE command facilitates the placement of programming comments within the body of a COMMAND FILE. It differs from the REMARK command, in that {characters} specified do *not* appear on the screen or printer when the program is executed.

EXAMPLE #1: A series of NOTES within the body of a COMMAND FILE:

Command File:

```
NOTE              File Name:  GLWORK.CMD
NOTE             Programmer:  Chester Holmes
NOTE        Revision Number:  3.2
NOTE Date of Last Revision:  11/25/85
{any statements}
```

An asterisk (*) serves the same purpose as "NOTE". It is often used by programmers to disable a line (or lines) of commands for testing purposes.

Operators

FULL FORM: (Does not apply)

PURPOSE: Used for manipulating data

OVERVIEW: There are four basic types of operators in **H & D Base**:

1.) ARITHMETIC OPERATORS:

There are four "Arithmetic Operators":

+	:Addition
—	:Subtraction
*	:Multiplication
/	:Division

Parentheses ( ) are used for grouping.

Order of precedence:

First:	Parentheses, functions
Second:	* , /
Third:	+ , —
Fourth:	Relations

2.) RELATIONAL OPERATORS

There are seven "Relational Operators":

<	:Less than
>	:Greater than
=	:Equal to
< >	:Not equal to
< =	:Less than or equal to
> =	:Greater than or equal to
$	:Substring is within string

3.) LOGICAL OPERATORS

There are three "Logical Operators":

.NOT.	:Boolean "NOT"
.AND.	:Boolean "AND"
.OR.	:Boolean "OR"

Parentheses ( ) are used for grouping.

Order of precedence:

First:	.NOT.
Second:	.AND.
Third:	.OR.

4.) STRING OPERATORS

There are two "String Operators":

+	:String concatenation
—	:String concatenation (Trailing blanks of first string moved to end of blanks in second string)

Parentheses ( ) are used for grouping.

Order of precedence:

First:	Parentheses, functions
Second:	Relations, $(substring operator)
Third:	+ , — (Concatenation)

Pack

FULL FORM: PACK

PURPOSE: Used to eliminate records marked for deletion

OVERVIEW: The PACK command purges from the USE file all records marked for deletion by the DELETE command. Once it has been issued, *there is no way to bring back the deleted records*.

If an INDEX file (or files) is being used in association with the USE file at the time the PACK is instituted, they will be reindexed. Any INDEX file *not* in USE will *not* be reindexed.

Records which have been marked for elimination with the DELETE command do *not* necessarily have to be "dumped" through a PACK. At times, it may be beneficial to include them in the database. Remember, records marked for deletion are *not* COPIED, APPENDED, or SORTED, but they are COUNTED.

If it becomes important to know whether or not the record being processed has been marked for deletion, the following series of commands may be helpful:

```
LOCATE FOR {exp}
DO WHILE .NOT. EOF
   IF .NOT. *
   [COMMANDS]
   ENDIF
   CONTINUE
ENDDO
```

They will skip over a record that has been deleted and continue processing with the next record. Note: Another method of avoiding the inadvertant use of deleted records is to SET DELETED "ON".

The COPY command can be used to achieve the same purpose as a PACK. When an old file is copied to a new file, DELETED records are *not* transferred.

Note: The PACK command is one of the most taxing functions the program performs. It is important, therefore, that *nothing* interrupt its execution. Of special concern is the "Escape" key. *Never press it during a PACK because data transmission errors will undoubtedly occur.*

Quit

FULL FORM: QUIT

PURPOSE: Used to terminate the operation of **H & D Base**

OVERVIEW: The QUIT command, when issued from either the COMMAND or PROGRAM LEVELS of the program, closes *all* files, clears all memory, and returns control to the Atari DESKTOP.

Read

FULL FORM: READ

PURPOSE: Used for accepting data into GET commands

OVERVIEW: When a READ command is encountered within the body of a COMMAND FILE, the program enters the full-screen mode to allow the entry and/or edit of variables which had previously been identified and displayed by "@" commands with GET phrases. While in this full-screen mode, the cursor can be moved to any of the GET variables. Changes made to those variables on the screen are entered into the appropriate database fields or memory variables.

These are the cursor and control keys which will work with the READ command:

1.) {UP ARROW} or {CTRL-E}
Moves the cursor one field up
2.) {DOWN ARROW} or {CTRL-X}
Moves the cursor one field down
3.) {LEFT ARROW} or {CTRL-S} or {BACKSPACE}
Moves the cursor one character to the left
4.) {RIGHT ARROW} or {CTRL-D}
Moves the cursor one character to the right
5.) {CTRL-W}
Exit - Save data (including all changes)
6.) {CTRL-Q}
Exit - Save data (except changes to data field variables)

The READ command can only be used when the FORMAT has been SET to the SCREEN. The sequence usually begins with the issue of an ERASE com mand (which clears the screen), followed by a series of "@...SAY ...GET" commands which serve to format the screen. The READ command, which con-

Read, cont.

cludes the sequence, allows information to be entered or edited in the GET statements. If no READ command appears, the GETS will not be executed and will remain "open."

Variables to be used with "@" commands and edited using READS must be either data field variables or character string memory variables. Memory variables must be pre-defined before the "@" command is issued. Note: There will be times when you will want to create a memory variable with nothing in it. To do so, use the "BLANK STRING" function.

If another series of "@" commands is issued *after* a READ command, they must be followed with *their own* READ. This READ will place the cursor on the first GET variable *following the last READ*.

EXAMPLE #1: A "@...SAY...GET...READ" sequences in the same COMMAND FILE:

```
ERASE
STORE "EDIT MODE " TO MHead1
STORE "---------------" TO MHead2
@ 3,40-((LEN(MHead1))/2) SAY MHead1
@ 4,40-((LEN(MHead2))/2) SAY MHead2
STORE Y TO MYesNo
DO WHILE MYesNo
  STORE SPACE(30) TO MName
  @ 7,15 SAY "Name to Edit: " GET MName
  READ
  FIND &MName
  @ 7,15
  IF .NOT. # = 0 @ 7,15 SAY " Name: " GET Name
    @ 9,15 SAY " Address: " GET Address
    @ 11,15 SAY " City: " GET City
    @ 11,48 SAY "State: " GET State PICTURE "!!"
    @ 11,59 SAY "Zip: " GET Zip PICTURE "99999"
    @ 13,15 SAY "Account Number: " GET Acct:Num PICTURE "99999"
    READ
  ENDIF
  @ 17,15 SAY "Continue? (Y/N) " GET MYesNo
  READ
  @ 17,15
ENDDO
```

If the "SET FORMAT TO {format file}" command has been issued, READ will cause all of the "@" commands in the format file to be executed, thus formatting the screen, allowing editing of all GET variables. Notice that this technique is a tailorable substitute for the EDIT command when at the COMMAND LEVEL.

Recall

FULL FORM: RECALL [{scope}] [FOR {exp}] [WHILE{exp}]

PURPOSE: Used to "undelete" records marked for deletion

OVERVIEW: The RECALL command, when issued in relation to a record (or records) in the USE file, removes the record (or records) from the "marked for deletion" status.

EXAMPLE #1: The DELETION and RECALL of records:

```
USE Ledger ok
DELETE ALL
00015 DELETIONS ok
RECALL RECORD 3
1 RECORD(S) RECALLED ok
GOTO RECORD 3 ok
? #
.F.
RECALL ALL FOR Acct:Num< >48376
00013 RECALL(S)
LIST NEXT 5

00001       20029        5.95        89.38
00002       58938       52.27      2898.86
00003       10003      267.92     10826.62
00004       99288       28.26      7822.22
00005      *48376        9.37       728.98
```

Reindex

FULL FORM: REINDEX

PURPOSE: Used to re-create an INDEX file

OVERVIEW: The REINDEX command is exactly the same as the "INDEX on {exp} TO {index file name}", with one notable exception. With REINDEX, the {exp} (or "Key") and the name of the "TO {file}" need not be entered. REINDEX uses the "Keys" from the INDEX files currently in use and completely rebuilds the "TO {file/s}".

EXAMPLE #1: The use of the REINDEX Command:

```
USE PEOPLE INDEX NAMEIND ok
REINDEX
```

Release

FULL FORM: RELEASE [{memvar list}]
[ALL]
[ALL LIKE {skeleton}]
[ALL EXCEPT {skeleton}]

PURPOSE: Used to get rid of unwanted memory variables

OVERVIEW: The RELEASE command eliminates (or "dumps") memory variables in order to free space for the creation of new memory variables.

Using the various forms of the command, users may RELEASE *all* memory variables, specific memory variables, or memory variables included (or not included) in a particular {skeleton}.

EXAMPLE #1: Typical RELEASE statements:

```
DISPLAY MEMORY

MEMDUCK    (C)  Donald
MLEGHORN   (C)  Foghorn
MEMGHOST   (C)  Casper
MRUBBLE    (C)  Barney
MEMBUNNY   (C)  Bugs
MJETSON    (C)  George
**TOTAL         06 VARIABLES USED   36 BYTES USED ok

RELEASE MemGhost,MJetson ok
DISPLAY MEMORY

MEMDUCK    (C)  Donald
MLEGHORN   (C)  Foghorn
MRUBBLE    (C)  Barney
MEMBUNNY   (C)  Bugs
**TOTAL         04 VARIABLES USED   24 BYTES USED ok
```

Remark

FULL FORM: REMARK [any characters]

PURPOSE: Used in the body of a COMMAND FILE to register a comment which will appear on the screen when the program is "run"

OVERVIEW: The REMARK command facilitates the placement of text in a COMMAND FILE which will be displayed on the output device when the program is "run." Note: It can also be issued from the COMMAND LEVEL of the program, but serves little purpose.

A blank space *must* appear between the command and the first character in the string. The string can be no longer than 254 characters. There is no limitation on the type of characters which may be used in [any characters].

The NOTE or "*" commands should be used to put comments in a COMMAND FILE which will *not* be displayed on the screen when the program is "run".

EXAMPLE #1: REMARKS at the beginning of a COMMAND FILE:

Command File:

```
REMARK H & D BASE TEMPLATES
REMARK Written by Chester Holmes & Oliver Duckworth
REMARK © Copyright 1986, All Rights Reserved
?
?
{commands}
```

Rename

FULL FORM: RENAME {original file name} TO {new file name}

PURPOSE: Used to give a file a new name

OVERVIEW: The RENAME command facilitates the change of a file's name. Three-character "trailers" (file type such as ".DAT", "NDX", etc.) must be used unless both the {original file} and {new} file are of the type ".DAT".

EXAMPLE #1:

```
RENAME ACCTPAY1 TO ACCTPAY2 ok
```

EXAMPLE #2:

```
RENAME B:REPORT.FRM TO REPORT.BAK ok
```

Replace

FULL FORM: REPLACE [{scope}] {field} WITH {exp}
[,{field} WITH {exp}] [FOR {exp}] [WHILE {exp}]

PURPOSE: Used for the rapid alteration of data in a database

OVERVIEW: This command is used to REPLACE the contents of specified data fields of the file in USE with the new data. One or more fields can be REPLACED using the same command.

If no {scope} is supplied in the command, REPLACE acts only on the current record.

EXAMPLE #1: The REPLACEMENT of the contents of two fields in all records of a database:

```
USE Ledger ok
LIST ok

00001      20029        5.95        89.38
00002      58938       52.27      2898.86
00003      10003      267.92     10826.62
00004      99288       28.26      7822.22
00005      48376        9.37       728.98
00006      00012      356.90     12872.90
00007      76926     8291.92    102930.83
00008      20835       26.00       372.50
00009      02678      105.83      2039.46
00010      83789     3893.29     68278.28
00011      10235        4.29        83.83
00012      38769     1083.36      8327.73
00013      69361      356.01      9367.62
00014      20835       10.50       175.00
00015      82919        2.98        29.95  ok

REPLACE ALL Balance WITH Cur:Chgs + Balance,Cur:Chgs
WITH 0
00015 REPLACEMENT(S)
LIST

00001      20029       00.00        95.33
00002      58938       00.00      2951.13
00003      10003       00.00     11094.54
```

```
00004     99288      00.00       7850.48
00005     48376      00.00        738.35
00006     00012      00.00      13229.80
00007     76926      00.00     111222.75
00008     20835      00.00        398.50
00009     02678      00.00       2145.29
00010     83789      00.00      72171.57
00011     10235      00.00         88.12
00012     38769      00.00       9411.09
00013     69361      00.00       9723.63
00014     20835      00.00        185.50
00015     82919      00.00         32.93 ok
```

If a REPLACE is executed on an INDEX "Key" and the INDEX is in USE, the index file will be properly adjusted. Any such INDEXES *not* in USE when the REPLACE is executed will *not* be updated. When a REPLACE is done on an INDEX "Key", the record which has been altered will most likely move to a different position in the file. The new "next record" will, therefore, not be the same as the old "next record".

"Keys" should not be REPLACED with "NEXT *n*" as the {scope}.

The REPLACE command can be used in relation to field variables, but not memory variables.

Note: If you are using PRIMARY and SECONDARY databases, you can REPLACE a field only in the currently SELECTED database.

Report

FULL FORM: REPORT [FORM {form file}] [{scope}] [TO PRINT] [PLAIN] [FOR{exp}] [WHILE{exp}]

PURPOSE: Used to generate a report from the information in a database

OVERVIEW: **H & D Base** gives users the ability to generate reports (either on the screen or on paper) which display data in a defined manner. The REPORT command is used for this purpose.

REPORT will list the data from the USE database in columns, and give users the option of including a header at the top of each page of the REPORT, headers above each column of data, and totaled numeric fields (all or part). An optional second header can be added using the "SET HEADER" command. The page number will always be listed in the upper left corner of the page along with a date.

The data displayed in each column of a REPORT is not necessarily just the information from a particular field of a record (although that is the most common use). The data is actually generated from an {expression} which can include not only field names, but memory variables and literals as well.

The "FOR {exp}" phrase allows only the records which meet the conditions of the {exp} to be REPORTED. The TO PRINT phrase sends the REPORT to the printer as well as the screen. The {scope} of the REPORT defaults to ALL unless otherwise specified.

The optional PLAIN command creates a REPORT that can be inserted into a text file generated by a word processor. When included in the statement, the command causes page numbers and the date at the top of each page in the report to be suppressed. Page headings are inserted into the **H & D Base** report only at the beginning of the report. The ejection of a page at the beginning of the REPORT can be suppressed with the SET EJECT "OFF" command.

The process of formatting a particular REPORT need only be done one time. Once all the specifications have been entered, **H & D Base** stores them in a "FORM" file (type ".FRM") on the data disk, and allows you to use that format over and over again. Include the optional "FORM {file name}" clause to do so.

Note: After a "FORM" file has been created and stored to disk, the "FORM" *can* be altered using the MODIFY STRUCTURE command.

The prompts below are listed as they appear when the format of a REPORT is being entered. The number opposite each prompt corresponds with an explanation of that part of the process.

REPORT FORM

```
1.) USE {database filename}
2.) REPORT

3.) REPORT FORM NAME:
4.) DEFAULT OPTIONS: LEFT MARGIN (M) = 8, LINES/PAGE (L) = 57,
    PAGE WIDTH (W) = 80
    ENTER OPTIONS:
5.) DO YOU WANT A PAGE HEADING (Y/N)?
      PAGE HEADING CAN BE 60 CHARACTERS OR LESS
      ENTER HEADING:
6.) DO YOU WANT TO DOUBLE SPACE BETWEEN LINES (Y/N)?
7.) DO YOU WANT TO TOTAL SPECIFIED NUMERIC FIELDS (Y/N)?
      DO YOU WANT TO SUBTOTAL SPECIFIED NUMERIC FIELDS (Y/N)?
      ENTER FIELD OR EXPRESSION TO SUBTOTAL ON (60 CHAR. MAX)
        :
      DO YOU WANT ONLY A SUMMARY REPORT (Y/N)?
      DO YOU WANT EACH SUBTOTAL ON A SEPARATE PAGE (Y/N)?
      ENTER A HEADING FOR SUBTOTALS (40 CHARACTERS OR LESS)
        :
8.) FIELD INFORMATION:
      COL   WIDTH,CONTENTS
      01
      ENTER HEADING:
      DO YOU WANT TO TOTAL THIS FIELD (Y/N)?
```

Report, cont.

EXPLANATIONS: 1.) The USE File

The file the REPORT is to be generated on must be in USE before the REPORT command is issued. Any valid **H & D Base** database file (type ".DAT") can be used.

2.) The REPORT Command

Entered without any qualifiers, "REPORT" will cause a new format to be created using as a basis the information from the database file in USE. If no database file had been previously specified, a prompt will appear asking for the name of a database file.

It is not necessary to specify a USE file if the optional "FORM {form file}" clause is included in the REPORT command. Its name is included in the {form file}.

3.) Report Form Names

The parameters for naming REPORT FORMS are the same for all **H & D Base** files; e.g., they must start with alpha characters, can be up to 8 characters long, and cannot include any colons or blank spaces.

Each FORM will be stored as a file with a ".FRM" trailer.

4.) Default Options

The defaults for the printing of a REPORT are:

Left Margin (M) = 8
Lines/Page (L) = 57
Page Width (W) = 80

If you want to use those defaults, just press {CR}. To change them, use this form:

M = {left margin}, L = {lines/page}, W = {page width} {CR}

For example:

```
ENTER OPTIONS: M = 10, L = 50, W = 70 {CR}
```

Note: Page Width is used for centering page headings only.

5.) Page Headings

A "Page Heading" is a title which will appear centered at the top of *every* page of the REPORT. Its maximum length is 80 characters.

6.) Double Spaced Reports

REPORTS will be "Double Spaced" if a "Y" for "YES" is registered to this prompt.

7.) Totals

Totals for *specified* columns of data which is numeric in nature will appear at the very end of the REPORT if you so choose with this option. Specify which columns you desire totals for in #9 (below).

If (and only if) an affirmative response is registered, you will be asked if you would like to subtotal specified numeric fields.

7.1) Subtotals

Subtotals can be generated for records *of like data* in a particular field of a database. Headings can be specified not only for columns, but for subtotal groups as well. The subtotal heading *includes* the "Key" of the like data. Column totals will automatically appear at the end of the REPORT. Important note: The USE file should be *indexed* on the field to be subtotaled and that index should be in USE when the REPORT command is initially issued.

If an "affirmative" response is registered to the subtotals prompt, these additional PROMPTS will appear:

7.11) ENTER FIELD OR EXPRESSION TO SUBTOTAL ON (60 CHAR. MAX.):

The name of the field to be subtotaled.

7.12) DO YOU WANT ONLY A SUMMARY REPORT ONLY (Y/N)?

You may choose to exclude the actual data from the file and list *only* the headings, subtotals, and totals. This is referred to as a "Summary Report."

7.13) DO YOU WANT EACH SUBTOTAL ON A SEPARATE PAGE (Y/N)?

A page eject at the end of each group of like records can be specified at this point.

7.14) ENTER A HEADING FOR SUBTOTALS (40 CHARACTERS MAX):

This heading will appear at the top of *each group* of like records. The contents of the field of those records will appear following the heading.

Example:

Orders for part number 11187

"Orders for part number" is the heading entered.

The program defaults to two decimal places. The "SET DECIMAL TO" command can be used to change the number of decimal places.

8.) Field Information

You must specify the width and content {expression} of each column to be included in the REPORT. They must be entered in this form:

{width},{exp} {CR}.

The width registered here can be smaller than, equal to, or larger than the actual width generated by the {expression}. If it is too small, data will be "wrapped around" to the next line. If it is too big, trailing blanks will be displayed (a waste of space).

The contents of the column is an {expression} which may be comprised of fields from the USE database, memory variables, and literals. Up to 24 columns can be entered.

A heading can be placed at the top of each column. If *no* column heading is desired for this particular column, simply press {CR}.

There are a few special characters which may be used in column headings. A semicolon (;) will break the heading at the semicolon and resume the display on the next line. If a heading is too long to fit within the number of spaces allowed for it, it will be broken at the last blank (if possible) and resumed on the next line. If the title is preceded with a "<" the title will be left-justified in the column. Likewise, a ">" will right-justify the title.

The "DO YOU WANT TO TOTAL THIS FIELD (Y/N)?" prompt will only appear if an "affirmative" answer was registered to the "DO YOU WANT TO TOTAL SPECIFIED NUMBERIC FIELDS (Y/N)" prompt (#7) *and* the column being defined is numeric in nature. Totals for each column specified will appear at the very end of that column.

When a "carriage return" (alone) is issued instead of a "{width} and {expression}", the format process is terminated. The REPORT will display on the screen (and printer if specified), and the format will be saved under the name specified when the process was begun (trailer ".FRM").

Before a REPORT is printed, a page is ejected. This may be suppressed with the SET EJECT "OFF" command. The SET HEADING TO command allows *an additional* page heading to be added to the top of every page of a REPORT during a single "run" of the program. (The heading must be set each time a new **H & D Base** run is initiated.) The additional heading will appear *above* the normal page heading. The same is true for

Report, cont.

the SET DATE TO command. The date of the report may be changed or omitted by use of this command. See the SET command for more information.

EXAMPLE #1: A simple REPORT:

```
USE Names ok
LIST

00001   HOWARD, ROY          2387 W. Benedict
   Miami      FL   27865  10235
00002   THOMAS, BRENDA    87 Laurel - Apt. #5
   Chicago    IL   48976  76926
00003   MURPHY, VINCE        987-B Hallowell
   Phoenix    AZ   87356  58938
00004   ALIMBO, JENNY        9376 Main
   Seattle    WA   99876  38769
00005   HAMILTON, JOHN       728 Downey Place
   Dallas     TX   49039  02678
00006   HORMAN, EILEEN     2 E. Plaskett Ct.
   Phoenix    AZ   87356  58397
00007   CUCUK, CHERYL        746 Manhassett Lane
   San Jose   CA   94687  10003
00008   ZACHRY, DAVE         P.O. Box 93876
   Reno       NV   93713  20835
00009   FAORO, ASHLEY        8467 Quinton
   Denver     CO   82076  69361
00010   MADDEN, KRIS         24 S. Broadway
   Boston     MA   02815  48376   ok

REPORT
REPORT FORM NAME: BasicRep
DEFAULT OPTIONS: LEFT MARGIN (M) = 8, LINES/PAGE (L) = 57,
PAGE WIDTH (W) = 80
ENTER OPTIONS: {CR}
DO YOU WANT A PAGE HEADING (Y/N)? Y
    PAGE HEADING CAN BE 60 CHARACTERS OR LESS
    ENTER HEADING: Current Customers
DO YOU WANT TO DOUBLE SPACE BETWEEN LINES (Y/N)? N
DO YOU WANT TO TOTAL SPECIFIED NUMERIC FIELDS (Y/N)? N
FIELD INFORMATION:
COL      WIDTH,CONTENTS
01       20,Name
 ENTER HEADING:  Customer
02       10,City
 ENTER HEADING:  City
03       5,State
 ENTER HEADING:  State
04       5,Zip:Code
 ENTER HEADING:  Zip
05       7,Acct:Num
 ENTER HEADING:  Account
06       {CR}

PAGE NO. 00001
12/15/85
```

```
                      Current Customers
      Customer            City       State   Zip      Account
HOWARD, ROY          Miami       FL      27865     10235
THOMAS, BRENDA       Chicago     IL      48976     76926
MURPHY, VINCE        Phoenix     AZ      87356     58938
ALIMBO, JENNY        Seattle     WA      99876     38769
HAMILTON, JOHN       Dallas      TX      49039     02678
HORMAN, EILEEN       Phoenix     AZ      87356     58397
CUCUK, CHERYL        San Jose    CA      94687     10003
ZACHRY, DAVE         Reno        NV      93713     20835
FAORO, ASHLEY        Denver      CO      82076     69361
MADDEN, KRIS         Boston      MA      02815     48376       ok
```

EXAMPLE #2: A REPORT with Totals and Subtotals:

```
USE Ledger ok
INDEX ON Part:Num TO PartInd
12 RECORD(S) INDEXED ok
SET INDEX TO PartInd ok
LIST ok

00002     McKINLEY'S       4    00194     .89     3.56
00005     MIKE'S AUTO      1    00194     .89      .89
00008     RESPONSIVE     143    00194     .89   172.66
00001     STANDARD        10    20029    5.95    59.50
00007     GUARANTEE        7    20029    5.95    41.65
00011     COASTAL        104    28947     .14    14.56
00004     PROGRESS        14    38372    1.39    19.46
00010     FARMINGTON      30    38372    1.39    41.70
00003     VALLEY          26    59839    7.45   193.70
00009     BARTON'S        66    68269    1.79   118.14
00012     UPTOWN          48    68269    1.79    85.92
00006     AJAX            18    73020     .10     1.80      ok

SET HEADING TO "Acme Distributors" ok
REPORT

REPORT FORM NAME: SubRep
DEFAULT OPTIONS: LEFT MARGIN (M) = 8, LINES/PAGE (L) = 57,
PAGE WIDTH (W) = 80
ENTER OPTIONS: {CR}
DO YOU WANT A PAGE HEADING (Y/N)? Y
  PAGE HEADING CAN BE 60 CHARACTERS OR LESS
  ENTER HEADING: Part Order Summary
DO YOU WANT TO DOUBLE SPACE BETWEEN LINES (Y/N)? N
DO YOU WANT TO TOTAL SPECIFIED NUMERIC FIELDS (Y/N)? Y
  DO YOU WANT TO SUBTOTAL SPECIFIED NUMERIC FIELDS (Y/N)? Y
    ENTER FIELD OR EXPRESSION TO SUBTOTAL ON (60 CHAR.
MAX)
        :Part:Num
    DO YOU WANT ONLY A SUMMARY REPORT (Y/N)? N
    DO YOU WANT EACH SUBTOTAL ON A SEPARATE PAGE (Y/N)? N
    ENTER A HEADING FOR SUBTOTALS (40 CHARACTERS OR LESS)
        :Orders for part number
FIELD INFORMATION:
COL     WIDTH,CONTENTS
```

Report, cont.

```
01          15,Customer
ENTER HEADING: <Customer Name
02          8,Quantity
ENTER HEADING: >Quantity;>Ordered
DO YOU WANT TO TOTAL THIS FIELD (Y/N)? Y
03 {CR}

PAGE NO. 00001
12/25/85
                          Acme Distributors
                         Part Order Summary

      Customer Name                              Quantity
                                                 Ordered

      Orders for part number 00194
      McKINLEY'S                                        4
      MIKE'S AUTO                                       1
      RESPONSIVE                                      143
      **SUBTOTAL**
                                                      148

      Orders for part number 20029
      STANDARD                                         10
      GUARANTEE                                         7
      **SUBTOTAL**
                                                       17

      Orders for part number 28947
      COASTAL                                         104
      **SUBTOTAL**
                                                      104

      Orders for part number 38372
      FARMINGTON                                       30
      PROGRESS                                         14
      **SUBTOTAL**
                                                       44

      Orders for part number 59839
      VALLEY                                           26
      **SUBTOTAL**
                                                       26

      Orders for part number 68269
      BARTON'S                                         66
      UPTOWN                                           48
      **SUBTOTAL**
                                                      114

      Orders for part number 73020
      AJAX                                             18
      **SUBTOTAL**
                                                       18
      **TOTAL**
                                                      471         ok
```

Restore

FULL FORM: RESTORE FROM {file} [ADDITIVE]

PURPOSE: Used to remember saved variables

OVERVIEW: RESTORE performs the opposite function of the SAVE command. A set of memory variables stored on a disk in a file of the type ".MEM" can be read back into the system using the RESTORE command. All memory variables which were defined previous to the execution of the RESTORE command are deleted when it is issued.

If the ADDITIVE phrase is included, memory variables already defined are *not* released, but are appended with those from the ".MEM" file.

EXAMPLE #1: Memory variables SAVED to a file then RESTORED:

```
DISPLAY MEMO

MSTRING     (C)     A1B2C3D4E5
MNUBER      (N)     12345
MLOGIC      (L)     .T.
**TOTAL**           03 VARIABLES USED     00027 BYTES USED ok

SAVE TO MFile ok
RELEASE ALL ok
DISPLAY MEMORY
**TOTAL**           00 VARIABLES USED     00000 BYTES USED ok

RESTORE FROM MFile ok
DISPLAY MEMORY
MSTRING     (C)     A1B2C3D4E5
MNUBER      (N)     12345
MLOGIC      (L)     .T.
**TOTAL**           03 VARIABLES USED     00027 BYTES USED ok
```

Return

FULL FORM: RETURN

PURPOSE: Used to terminate a COMMAND FILE and return to the calling file or COMMAND LEVEL

OVERVIEW: The RETURN command, when encountered by the program within the body of a COMMAND FILE, causes control to be returned either to the COMMAND FILE which called it or to the COMMAND LEVEL (if the file was called from the keyboard).

The RETURN command is equivalent to encountering the end of a COMMAND FILE.

It is a good programming practice to include a RETURN command as the last executable line of any COMMAND FILE.

Save

FULL FORM: SAVE TO {file} [ALL LIKE {skeleton}]

PURPOSE: Used to write current memory variables to a file for future use

OVERVIEW: The SAVE command stores all currently defined memory variables to a file of the type ".MEM". These memory variables may be returned to the active memory of the computer using the RESTORE command.

The command SAVES all current memory variables unless the {skeleton} clause is included. With it, you can specify a certain group of memory variables to be stored (see {skeleton} in "Symbol Definitions" at the beginning of this section).

EXAMPLE #1: SAVING and RESTORING memory variables:

```
STORE "A1B2C3D4E5" TO MString ok
STORE 12345 TO MNumber ok
STORE Y TO MLogic ok
DISPLAY MEMO

MSTRING      (C)    A1B2C3D4E5
MNUBER       (N)    12345
MLOGIC       (L)    .T.
**TOTAL**           03 VARIABLES USED       00027 BYTES USED ok

SAVE TO MFile ok
CLEAR ok
DISPLAY MEMORY

**TOTAL**           00 VARIABLES USED       00000 BYTES USED ok

RESTORE FROM MFile ok
DISPLAY MEMORY

MSTRING      (C)    A1B2C3D4E5
MNUBER       (N)    12345
MLOGIC       (L)    .T.
**TOTAL**           03 VARIABLES USED       00027 BYTES USED ok
```

Select

FULL FORM: SELECT [PRIMARY]
[SECONDARY]

PURPOSE: Used to switch database work areas

OVERVIEW: **H & D Base** allows users to select one of two different database areas to work in. This facilitates operations such as the update of one database with the data of another and the comparison of the data in two databases (to name a few). The two areas are called the "PRIMARY" and "SECONDARY" work areas. The SELECT command serves as a "toggle" between them.

When **H & D Base** is first entered, the PRIMARY area is active. All work is done in this PRIMARY area unless a SELECT SECONDARY instruction is issued. If (and when) it is selected, all work is conducted in the SECONDARY area (and so forth). A different database may be USED in each of the areas. This permits the concurrent usage of two databases at once. **H & D Base** commands that cause movement of the database (i.e. GOTO, SKIP, REPORT, SORT, COPY, LIST, DISPLAY and others) affect *only the currently selected database*. The REPLACE command will only affect variables in the currently selected database. The DISPLAY STRUCTURE command will display the structure of the currently selected database only.

EXAMPLE #1: USING PRIMARY and SECONDARY databases and performing miscellaneous commands on them:

Command File:

```
SELECT PRIMARY
USE Names
GOTO TOP
SELECT SECONDARY
USE Ledger
LOCATE FOR S.Acct:Num = P.Acct:Num
```

```
SELECT PRIMARY
? Name,S.Acct:Num,Balance
```

This command file displays a client's name, balance, and account number -- information pulled from two different databases.

The SET LINKAGE "ON" command allows all *sequential* commands (those that have a {scope} parameter) to perform positioning on both the PRIMARY and the SECONDARY databases. (Refer to the SET LINKAGE command in this *Reference* section).

When both database areas have databases in USE, an expression entered may use data field variables from *either* area. If the field name in both regions are the same for a desired variable, the variable must be prefixed with a "P." or "S." to denote which database it is to come from.

Set

FULL FORMS:	1.) SET {parameter} [ON] [OFF]
	2.) SET {parameter} TO {option}

PURPOSE: Used for changing the configuration of **H & D Base**

OVERVIEW: **H & D Base** provides a number of "SET" commands which control how the program interacts with your system. There are two types of SET commands. The first are merely toggle switches which register a default of either "ON" or "OFF". The second allows you to enter specific values. *All* SET commands can be quickly and easily changed from either the COMMAND LEVEL of the program or from within a COMMAND FILE.

Each of the SET commands which serves as a simple "ON/OFF" toggle, along with its default value and a brief description, is listed below.

SET BELL ON/OFF (Default = ON)

ON: Enables the bell which rings when a field has been filled during various functions or when data of the wrong type is entered
OFF: Disables the bell
NOTE: The volume of the bell can be altered from the Control Panel of the Atari DESKTOP

SET CARRY ON/OFF (Default = OFF)

ON: Carries data from the record just entered forward to the new record when in APPEND
OFF: Does not carry data forward -- shows just blank record
NOTE: Used for entering a large number of records which contain the same information in many of the fields

SET CONFIRM ON/OFF (Default = OFF)

ON: When a field has been filled in APPEND, EDIT, or READ, waits for a {CR} to be issued from the keyboard before continuing

OFF: Continues to next field automatically when current field is full

SET CONSOLE ON/OFF (Default = ON)

ON: All output is sent (echoed) to the screen

OFF: No output is sent to the screen

SET COLON ON/OFF (Default = ON)

ON: Uses colons on the screen to show the length of a field during AT...GET, APPEND, and EDIT functions

OFF: No colons appear

SET DELETED ON/OFF (Default = OFF)

ON: Records marked for deletion with an asterisk cannot be located with the FIND commands nor processed by any command that allows the NEXT phrase (such as LIST, LOCATE, COUNT, etc.)

OFF: Records marked for deletion *can* be located and displayed (but not copied or appended)

SET ECHO ON/OFF (Default = OFF)

ON: All commands which come from COMMAND FILE are also sent to the screen.

OFF: Commands from COMMAND FILE are *not* being sent to the screen.

NOTE: Very useful for debugging COMMAND FILES.

SET EJECT ON/OFF (Default = OFF)

ON: The REPORT command will eject a new page before beginning a new report

OFF: A new page will not be ejected

Set, cont.

SET ESCAPE ON/OFF	(Default = ON)	
	ON:	Allows a user to use the "Escape Key" {ESC} to break out of the execution of a COMMAND FILE
	OFF:	Disables the "Escape Key"
SET EXACT ON/OFF	(Default = OFF)	
	ON:	Requires that character strings match *exactly* (excluding trailing blanks) in expressions and the FIND command.
	OFF:	Allows matches to be made on the basis of the length of the second string.
	NOTE:	When "OFF", "ABC" is a match to "ABCDEFGH"
SET FORTH ON/OFF	(Default = OFF)	
	ON:	FORTH Program Language commands available.
	OFF:	FORTH Program Language commands not available.
SET LINKAGE ON/OFF	(Default = OFF)	
	ON:	Moves the record pointers in PRIMARY and SECON DARY files *simultaneously* (in increments) in association with commands that allow {scope} (LIST, REPORT, SUM, etc.).
	OFF:	Makes PRIMARY and SECONDARY record pointers work independent of one another.
SET PRINT ON/OFF	(Default = OFF)	
	ON:	All output is sent (echoed) to the printer
	OFF:	Output is not sent to the printer
SET RAW ON/OFF	(Default = OFF)	
	ON:	DISPLAYS and LISTS records without spaces between fields
	OFF:	DISPLAYS and LISTS records with an extra space between fields

SET TALK ON/OFF

(Default = ON)

ON: Displays the results of commands on the screen
OFF: Does *not* display the results of commands on the screen

The following SET commands require that you enter a certain value (as opposed to simply toggling between "ON" and "OFF"):

SET ALTERNATE TO {filename}

The SET ALTERNATE TO command is part of a two-step process for writing *everything* that is normally written on the screen to a disk file as well. The file will be stored with a ".TXT" trailer, indicating that it is a "Text File."

To establish a Text File, enter the following:

```
SET ALTERNATE TO {filename}
```

When you want *all* of the information appearing on the screen to also be sent to the specified Text File, enter:

```
SET ALTERNATE ON
```

To turn the flow of information "off," enter:

```
SET ALTERNATE OFF
```

The information in a text file can be edited, printed, etc. with the aid of a word processor or text editor.

SET COLOR TO {number of character color}

(Default = 0)

Text can be displayed in four different colors using the Atari ST series of computers. The colors, assuming that you have not altered them from the Control Panel of the DESKTOP, are white (the default color), black, red, and and green. They will appear as different shades of grey on a black and white monitor.

To change the color of the characters being typed or displayed on the screen, enter "SET COLOR TO" followed by one of the following numbers:

0 = White
1 = Red
2 = Green
3 = Black

All text entered or displayed from that point on will appear in the color you specified. Note: You may not be able to see black letters at all.

SET DATE TO {mm/dd/yy}

(Default = System Date)

The DATE function of **H & D Base** displays the date stored *in the Atari system* (set on the Control Panel of the DESKTOP) unless you override it with this command.

The date must be entered in the form shown (mm = month, dd = date, yy = year).

When you enter a date with this command, you are resetting the date stored in the Atari.

SET DECIMAL TO {*n*}

In REPORT, the default number of decimal places used in a numeric field is 2. You may change that number with this SET command.

SET DEFAULT TO {drive}

{Default = A}

Whenever you enter the name of a file to be used in relation to a certain command, **H & D Base**, assumes that the file resides on the disk in the default drive. You may indicate to the program that a file resides on a disk in an alternate drive by prefacing the file name with the letter which represents that drive followed by a colon.

For example:

USE People	(File located on default drive)
USE B:People	(File located on "B" drive)

The "SET DEFAULT" command allows you to choose the default disk drive that the program will access when searching for files which appear *without* a designated drive letter preceeding them.

A sample "SET DEFAULT" command:

```
SET DEFAULT TO C:
```

The trailing colon is optional.

SET FOLDER TO {folder name}

(Default = \ )

The SET FOLDER command opens a "folder" on the disk in the default drive which has been created either from the Atari DESKTOP or using the CREATE FOLDER command of **H & D Base**. A "folder" is a storage area for like files somewhat akin to a disk partition. If a particular FOLDER is specified using this SET command, the files specified in any command issued from **H & D Base** will be searched for, and written to the "folder." *Any file which is stored in a "folder" cannot be accessed unless that "folder" has been specified using this SET command.*

To terminate the use of a folder enter this command:

```
SET FOLDER TO  \
```

SET FORMAT TO [SCREEN] [PRINT] [{format file}]

(Default = SCREEN)

SCREEN:	Sends output of @ commands to the screen
PRINT:	Sends output of @ commands to the printer
{format file}:	Uses format previously created for APPEND, EDIT, and INSERT commands

SET HEADING TO {string}

This SET command adds a second heading to a REPORT. The {string} can be up to 60 characters long.

SET INDEX TO {index file} [,{index file},... {index file}]

This command opens up to seven index files for use in relation to the database file currently resident. It can be used to keep a number of INDEXES updated during functions like APPEND, EDIT, etc.

The first index file named is considered to be the "Master Index." The database file will be in indexed order according to it, and it is the one which will be used in all FINDS.

If an index file is already in use when this command is instituted, that index file will be closed before the new one is opened.

If you enter just "SET INDEX TO {CR}", all indexes will be released.

SET MARGIN TO {*nnn*}

This command can be used to change the left margin on the printer during the printing of a REPORT. The number entered can be no larger than "254".

Skip

FULL FORM: SKIP [+ {literal number}]
[—{literal number}]

PURPOSE: Used for moving the record pointer in a database

OVERVIEW: The SKIP command is used for moving the pointer forward and backward through the records of the USE database. Movements are executed relative to the current record position.

EXAMPLE #1: Miscellaneous applications of the SKIP command:

```
USE Names ok
LIST NEXT 5

00001  HOWARD, ROY        2387 W. Benedict
     Miami      FL  27865   10235
00002  THOMAS, BRENDA   87 Laurel - Apt. #5
     Chicago    IL   48976   76926
00003  MURPHY, VINCE      987-B Hallowell
     Phoenix    AZ  87356   58938
00004  ALIMBO, JENNY      9376 Main
     Seattle    WA  99876   38769
00005  HAMILTON, JOHN     728 Downey Place
     Dallas     TX  49039   02678  ok

GOTO TOP
SKIP ok
DISPLAY

00002  THOMAS, BRENDA   87 Laurel - Apt. #5
     Chicago    IL   48976   76926  ok

SKIP 3 ok
DISPLAY

00005  HAMILTON, JOHN     728 Downey Place
     Dallas     TX  49039   02678  ok
```

Sort

FULL FORM: SORT ON {expression} TO {file} [ASCENDING]
[DESCENDING]

PURPOSE: Used for generating a database that is SORTED on the contents of a particular field

OVERVIEW: When the SORT command is issued, a new database {file} is created using the records of the USE file. The records in the new file are placed in alphabetic or numeric order according to the contents of {expression}. That order can be either [ASCENDING] (the default) or [DESCENDING]. The USE file is not altered in any way, and remains "active" when the process of creating the new {file} has been completed.

Note: Records which have been DELETED are *not* transferred to the new {file}.

Because a completely new file is created using all (or nearly all) of the records of the original, make sure there is adequate room on your data disk for the new file before issuing the SORT command.

EXAMPLE #1: A simple SORT (DELETED records not transferred):

```
USE Names ok
DELETE ALL FOR Acct:Num>50000
00005 DELETIONS ok

LIST

00001   HOWARD, ROY            2387 W. Benedict
    Miami    FL   27865   10235
00002   *THOMAS, BRENDA        87 Laurel - Apt. #5
    Chicago   IL   48976   76926
00003   MURPHY, VINCE          987-B Hallowell
    Phoenix   AZ   87356   58938
00004   *ALIMBO, JENNY         9376 Main
    Seattle   WA   99876   38769
```

```
00005   *HAMILTON, JOHN        728 Downey Place
    Dallas      TX  49039  02678
00006    HORMAN, EILEEN        2 E. Plaskett Ct.
    Phoenix     AZ  87356  58397
00007   *CUCUK, CHERYL         746 Manhassett Lane
    San Jose    CA  94687  10003
00008   *ZACHRY, DAVE          P.O. Box 93876
    Reno        NV  93713  20835
00009    FAORO, ASHLEY         8467 Quinton
    Denver      CO  82076  69361
00010    MADDEN, KRIS          24 S. Broadway
    Boston      MA  02815  48376  ok

SORT ON Name TO Namesort
SORTING FILE - PLEASE WAIT ok
USE Namesort ok
LIST

00001    FAORO, ASHLEY         8467 Quinton
    Denver      CO  82076  69361
00002    HORMAN, EILEEN        2 E. Plaskett Ct.
    Phoenix     AZ  87356  58397
00003    HOWARD, ROY           2387 W. Benedict
    Miami       FL  27865  10235
00004    MADDEN, KRIS          24 S. Broadway
    Boston      MA  02815  48376
00005    MURPHY, VINCE         987-B Hallowell
    Phoenix     AZ  87356  58938  ok
```

If the "Key" {expression} is "character" in nature, the records of the SORTED file will appear in order *according to their respective ASCII code number*. In ASCII order, upper case letters are less significant than lower case letters. Therefore, in ascending order, "DUCKWORTH" will appear before "Duckworth".

All INDEXES files (type ".NDX") are automatically closed when the SORT command is issued.

You may SORT on any number of fields by using the command in this form:

```
SORT ON {field #1} + {field #2} + {field #3} TO {file}
```

{field #1} is the field of highest priority -- {field #3} the lowest.

EXAMPLE #2: SORTING a database on a number of "Keys":

```
USE Names ok
SORT ON City + State + Zip TO Names1
```

The INDEX command is very much like the SORT command. The major difference is that a SORT physically changes the order of the records in a file. In most cases, using INDEX will allow greater freedom and speed than SORT.

Store

FULL FORM: STORE {exp} TO {memvar}

PURPOSE: Used for placing values into memory variables

OVERVIEW: The STORE command computes the value of the {exp} then stores that value in a memory variable {memvar}. If the {memvar} does not exist at the time the command is issued, **H & D Base** will create it automatically.

You *cannot* STORE values to database field variables. Use the REPLACE command for that purpose.

The RELEASE command may be used to "dump" the contents of a {memvar} (or group of {memvars}).

EXAMPLE #1: Miscellaneous STORE commands:

```
STORE 5367 TO MNum ok
STORE "Holmes & Duckworth " TO MName ok
STORE "T" TO MYesNo ok
STORE MName + STR(MNum,5) TO MCombo ok
DISPLAY MEMORY

MNUM        (N)   5367
MNAME       (C)   Holmes & Duckworth
MYESNO      (L)   .T.
MCOMBO      (C)   Holmes & Duckworth5367
**TOTAL**         04 VARIABLES USED          00051 BYTES USED
```

Sum

FULL FORM: SUM [{scope}] {exp} [,{exp list}] [TO {memvar list}] [FOR {exp}] [WHILE {exp}]

PURPOSE: Used for totaling fields in a database

OVERVIEW: The SUM command is used for adding together numeric expressions (usually fields) of the database file in USE. Totals can be generated from {scope} records (default = ALL), or from selected records (using the optional [FOR {exp}] clause.

You may SUM up to five {expressions} with any one command using the form "SUM {exp},{exp},{exp},{exp},{exp}". The values will be returned to the screen in this form: {value} {value} {value} {value} {value}.

EXAMPLE #1: Miscellaneous SUM commands:

```
USE Ledger ok
LIST ok

00001  20029     5.95      89.38
00002  58938    52.27    2898.86
00003  10003   267.92   10826.62
00004  99288    28.26    7822.22
00005  48376     9.37     728.98
00006  00012   356.90   12872.90
00007  76926  8291.92  102930.83
00008  20835    26.00     372.50
00009  02678   105.83    2039.46
00010  83789  3893.29   68278.28
00011  10235     4.29      83.83
00012  38769  1083.36    8327.73
00013  69361   356.01    9367.62
00014  20835    10.50     175.00
00015  82919     2.98      29.95  ok

SUM Cur:chgs
14494.85 ok
SUM Balance
226844.16 ok
SUM Cur:chgs,Balance
```

```
14494.85            226844.16 ok
SUM Cur:chgs + Balance
241344.01 ok

LIST FOR Acct:Num > 80000
00004   99288      28.26      7822.22
00010   83789   3893.29    68278.28
00015   82919       2.98         29.95   ok

SUM Cur:chgs,Balance FOR Acct:Num > 80000
3924.53             76130.45 ok
```

If the optional TO clause is included in the statement, the SUM will be stored to the designated {memvar} as well as displayed on the screen. If the specified {memvar} doesn't exist when the SUM command is issued, it will automatically be created.

EXAMPLE #2: Storing SUMS to memory variables:

```
USE LEDGER ok
SUM Cur:chgs,Balance TO MCharge,MBalance ok
STORE MCharge + MBalance TO MTotal ok
DISPLAY MEMO

MCHARGE        (N)      14494.85
MBALANCE       (N)      226844.16
MTOTAL         (N)      241344.01
**TOTAL**               03 VARIABLES USED        00024 BYTES USED
```

Text

FULL FORM:

```
TEXT
   {text}
ENDTEXT
```

PURPOSE: Used for displaying blocks of text without the need of special formatting

OVERVIEW: When the TEXT statement appears in the body of a COMMAND FILE, *all* text between it and a concluding ENDTEXT statement is transferred as it appears directly to the screen or printer (depending on the SET status).

If an ampersand symbol (&) is included in the text, it will *not* be expanded (see "&"). If a "tilde" (~) is included at the end of a line of text, the beginning of the next line will be "moved up" to fill the blanks between it and the right margin. Use the TEXT command as a handy alternative to multiple "@...SAY"s or question marks (?).

EXAMPLE #1: Command File:

```
TEXT
   Four score and seven years ago, our Fathers brought forth upon this
   continent a new nation, conceived in liberty and dedicated to the propos-
   tion that all men are created equal.

   USE Names
   INDEX on Name to Nameind

   1!2@3#4%67*8(9)0
ENDTEXT
```

Total

FULL FORM: TOTAL ON {key} TO {database} (FIELDS {list}) (FOR {exp}) (WHILE {exp})

PURPOSE: Used for generating a database which includes subtotals for like records

OVERVIEW: The TOTAL command produces results which are very similar to the subtotal capability of the REPORT command. With the TOTAL command, however, subtotals are placed in a {database} instead of appearing on the screen or printer. The USE database must be either SORTED by the {key} or INDEXED on the {key}.

When issued, the TOTAL command will create a record in the TO {database} for each unique value of {key} appearing in the USE database. Fields of that record which are numeric in nature will be TOTALED.

If the TO {database} already exists when the TOTAL command is issued, its structure will be used to determine which fields of the USE database will be transferred. In other words, any field in the USE database which does not have a matching field in the TO database will not be transferred.

If the TO database does *not* exist when the TOTAL command is instituted, the structure from the USE database will be copied to the TO file. Records which have been combined from the information of two or more records of the original (USE) database will have TOTALED numeric fields and data from the last record of the like {key} in all other fields.

If the optional "FIELDS {list}" clause is included, only the numeric fields listed in it will be TOTALED.

Total, cont.

The TOTAL command can also be used to remove duplicate records from a database. Because non-numeric fields in "FIELDS {list}" are not totaled, duplicate records with the same {key} will be eliminated.

EXAMPLE #1: A photographer has received orders from a number of clients, some of which are for the same picture. How many of each picture to print?

```
USE Orders ok
DISPLAY STRUCTURE

STRUCTURE FOR FILE: ORDERS.DAT
INDEX IN USE: NONE
NUMBER OF RECORDS: 10
PRIMARY SELECTED

FIELD       NAME        TYPE      LENGTH       DEC
01          NAME        C            030
02          PICTURE     C            003
03          NUM:ORD     N            003
TOTAL BYTES:                       00037        ok

INDEX ON Picture TO Picind ok
SET INDEX TO Picind ok
LIST OFF

Montgomery Plaster             7        4
Ajax Fireworks                14        5
Keller Fireplace              14       11
Truffle's                     50        7
Frantz Muffler                88       14
Ingram Tool & Die            156        3
Cutler Television            254        2
Trophy Heaven                254       19
Autosound                    254        8
Mickey and Joe's             419        4       ok

USE Pictotal ok
DISPLAY STRU

STRUCTURE FOR FILE: PICTOTAL.DAT
INDEX IN USE: NONE
NUMBER OF RECORDS: 0
PRIMARY SELECTED

FIELD       NAME        TYPE      LENGTH       DEC
01          PICTURE     C            003
02          NUM:ORD     N            003

TOTAL BYTES:                       00007        ok

USE Orders INDEX Picind ok
```

```
TOTAL ON Picture TO Pictotal
7 RECORDS COPIED ok

USE Pictotal      ok
LIST OFF

   7             4
  14            16
  50             7
  88            14
 156             3
 254            29
 419             4 ok
```

Note condensation of orders for pictures #14 and #254.

Update

FULL FORM: UPDATE FROM {database} ON {key} [ADD {field list}] [RANDOM] [REPLACE [{field list}] [{field} WITH {from-field}]

PURPOSE: Used for modifying a database with data from another database

OVERVIEW: Using the UPDATE command, data in the USE database file can be modified using the information from an alternate {database}. When run SEQUENTIALLY (without specifying the [RANDOM] option), the program compares the contents of the {key} in the first record of the USE database file against the contents of the {key} in *first* record of the FROM database file. Whenever two {keys} match, the remainder of the the UPDATE command is executed on the record of the USE database file. It then compares the first record of the USE database with the *second* record in the FROM database. If the value of the {key} in the FROM database is greater than that of the USE database, the USE database advances one record. These "hopscotch" comparisons continue until the bottom of one of the files is reached. When the [RANDOM] option is included in the UPDATE statement, the program executes an internal FIND command to search for matches. Suffice it to say that the same results will be obtained regardless of which form you use.

Depending upon which way the UPDATE is done (SEQUENTIALLY or RANDOMLY), the following steps *must* be taken before the UPDATE command is actually issued:

SEQUENTIAL:

The USE file *must* be either SORTED or INDEXED (with the INDEX in USE) on the same {key} as the FROM database. The FROM file *must* be SORTED (not INDEXED) on the same {key} as the USE database.

RANDOM:

The {key} is assumed to be a single field in the FROM file, a field that can match INDEXES in the USE database. The FROM file does *not* have to be SORTED. The USE database *must* be INDEXED (not SORTED) on the {key}, and the INDEX *must* be in USE.

EXAMPLE #1: An application of the UPDATE command:

```
USE Autos ok
SORT ON Auto:Num TO Newautos ok
USE Newautos ok
DISPLAY STRUCTURE

STRUCTURE FOR NEWAUTOS.DAT
INDEX IN USE: NONE
NUMBER OF RECORDS: 4
PRIMARY SELECTED

FLD      NAMETYPE       LENGTH   DEC
 001     AUTO:NUMC      008
 002     ON:HANDN       005
 003     COSTN          010      002

TOTAL BYTES:                     00024              ok

LIST

00001    BUIK778811     2150.00
00002    DATS14985      1200.00
00003    PACK56789      1895.00
00004    STUD123414     2300.00

USE Invntory ok
INDEX ON Auto:Num TO Invidx
SET INDEX TO Invidx ok
DISPLAY STRU

STRUCTURE FOR INVNTORY
INDEX IN USE: INVIDX
    KEY: AUTO:NUM
NUMBER OF RECORDS: 6
PRIMARY SELECTED

FLD      NAMETYPE       LENGTH   DEC
 001     AUTOC          015
 002     AUTO:NUMC      008
 003     ON:HANDN       005
 004     COSTN          010      002

TOTAL BYTES:                     00037              ok

LIST
```

Update, cont.

```
00003     BUICK ELECTRA        BUIK7788    7     2150.00
00004     DATSUN 510           DATS1498    2     1075.00
00002     PACKARD DELUXE       PACK5678    5     1895.00
00001     STUDEBAKER STD.      STUD1234    9     2300.00
00005     TOYOTA SR5           TOYO7390    6     1675.00
00006     VOLKSWAGON BUG       VOLK1872    3     1550.00   ok

UPDATE ON Auto:Num FROM Newautos ADD On:Hand REPLACE Cost
ok
LIST

00003     BUICK ELECTRA        BUIK7788    18    2150.00
00004     DATSUN 510           DATS1498    7     1200.00
00002     PACKARD DELUXE       PACK5678    14    1895.00
00001     STUDEBAKER STD.      STUD1234    23    2300.00
00005     TOYOTA SR5           TOYO7390    6     1675.00
00006     VOLKSWAGON BUG       VOLK1872    3     1550.00   ok
```

Use

FULL FORM:

USE

USE [{database file}]

USE {database file} INDEX {index file} [,{index file}...]

PURPOSE: Used for opening a database for future operations

OVERVIEW: The USE command is used to tell **H & D Base** which {database file} is to be opened for potential manipulation. The {database file} must exist before the command is issued.

If a database file is already in USE when the command is executed (with or without a specified {database file}), it is automatically closed (unless it is re-chosen in which case it is closed and reopened). An INDEX file (or files) previously created for use with a particular database can be opened by entering the command in this form:

```
USE {database file} INDEX {index file} [,{index file}...]
```

Up to seven index files may be USED with a {database file} at the same time. The first {index file} specified is the "Master Index." All FINDS will use only this INDEX and the database will be in order according to the "Master Index." All of the {index files} specified in the statement will automatically be updated anytime their "Keys" are modified (by the APPEND, EDIT, REPLACE, READ, or BROWSE commands).

EXAMPLE #1: Applications of the USE command:

```
USE ok
USE People ok
USE Names INDEX Nameind ok
USE Names INDEX Nameind, Cityind, Zipind ok
```

Wait

FULL FORM: WAIT [TO {memvar}]

PURPOSE: Used to pause in a program, usually for input purposes

OVERVIEW: When the WAIT command is encountered in the body of a COMMAND FILE, a "WAITING" prompt appears on the screen and all operations come to a complete halt.

The depression of *any single key* on the keyboard causes the continuation of command execution. If the optional "TO {memvar}" clause is specified, the character which is represented by the pressed key will be entered into the {memory variable}.

If the key which is pressed is non-printable (i.e., RETURN, LINE FEED, or any other {CTRL} character, the value of the {memvar} is set to a blank.

Note: Because only one character can be entered, it is not necessary to press <CR>.

EXAMPLE #1: An application of the WAIT command:

```
WAIT TO MWait
WAITING 2 ok
? MWait
2 ok
```

In a COMMAND FILE:

```
USE Names ok
DO WHILE .NOT. EOF
   DISPLAY OFF
   ? " PRESS ANY KEY"
   WAIT
   SKIP
ENDDO
```

APPENDICES

Appendix A
The Nature of a Database

Definitions

Data - Significant items of information.

Database - A large collection of data.

Database Management System - A program for setting up and manipulating a database.

Relational - The ability of a Database Management System to manipulate two files concurrently.

How a Database is Organized

The information which has become such an important part of our daily lives is most commonly kept on FORMS. A FORM is a piece of paper upon which is printed a number of titles prompting us to fill-in information required by an individual or organization.

Each title, and the calibrated line which follows it, is a FIELD of information which the individual or organization feels important to store on you and the rest of the people which complete that particular FORM.

When you fill a FORM out, it becomes a RECORD of information which pertains solely to you. For example, when you first visited your doctor's office, he had you fill out a FORM. When finished, it was no longer a FORM, it was a RECORD which dealt exclusively with information on you, his patient.

RECORDS which have been filled out on the same FORM are grouped together in a storage facility called a FILE. Your doctor undoubtedly has a number of filing cabinets full of records pertaining to all of his clients. One of those cabinets contains the "Patient FILE" which he accesses in order to update your health RECORD, prepare a special mailing, or send out bills. Another might contain the "Prescription FILE" or the "Medicare FILE" (and so on).

The Nature of a Database, cont.

A Computer Database

The structure of a computer database uses all of the components discussed above. Users enter the specifications of a FORM on the screen which is comprised of a number of different FIELDS. There are no limitations on what the FIELDS can deal with (people, places, things, etc.), and few limitations on how large they can be.

Information is entered on this blank FORM, creating an individual RECORD with each completed FORM. The number of RECORDS a DATABASE user can enter is limited only by the size of each RECORD and the capacity of his or her disk drive. Information entry can be done in any number of sittings, in random order. The RECORDS are placed in a FILE on the disk drive which is identified by a special name defined by the user. Any number of FILES can be created.

Once entered, the information included on the RECORDS in this FILE can be manipulated in a virtually limitless number of ways. It can be listed (in random or sorted order), reviewed selectively, edited, appended, printed, and more and more. Various reports can also be generated using it.

A Database Management System

The creation and manipulation of a DATABASE in such a manner is accomplished through the use of a "Database Management System." No two "Database Management Systems" are the same. Some have rigid, non-alterable FORMS, while others simply don't provide the breadth of functions needed to process data properly.

H & D Base is a "Database Management System." It can do all of the things described above -- and much more. With **H & D Base** the commands which perform the manipulation of data can be grouped together into "programs," which, when run, perform the same task over and over again. **H & D Base** is also unique in that it is "Relational." It can process the data from *two* different FILES at the same time.

Appendix B Using Non-H & D Base Files

Introduction

If you have COMMAND FILES which were developed for use with "dBASE II" or data files which were generated by an alternate database management system, there is a good chance they will transfer to **H & D Base** (and vice-versa). The critical issue is whether or not the files can be set-up in either "SDF" (System Data Format) or "DELIMITED" formats.

SDF and DELIMITED Files

A file which is stored in "SDF" format has all of its fields on one line with a {CR} at the end of the line. Note: It is assumed that {CR}contains both a {CR} *and* a line feed. Example:

```
AJAX      2345 S. Maple      Akron      OH 48378      748.95{CR}
```

"DELIMITED" files have commas between each field and delimiters (usually single or double quotes) around all fields which are "Character" in nature. A {CR} also appears at the end of the line. Example:

```
"AJAX     ","2345 S. Maple      ","Akron ","OH","48378",    748.95{CR}
```

Receiving Files

To transfer a file to **H & D Base**, first make sure that file is in either the "SDF" or "DELIMITED" format. Note: All "dBASE II" COMMAND FILES are "SDF" in nature.

If the file comes from another Atari ST database program, just insert the disk on which it is stored into the drive. If the file comes from a database program on another computer, it must be sent to the Atari via a modem or direct connect cable (you must have communications software packages on both machines to do so).

If the file transferred is a "dBASE II" COMMAND FILE, chances are that it will load in and run with the "DO {command file}" statement. If not, check these possible causes:

1.) **H & D Base** uses a "tilde" (~) at the end of the first line of commands which are longer than one line. These tildes must be replaced with semi-colons (;).

2.) There can be no embedded {CTRL} characters (except {CR}'s)

3.) The first word in any **H & D Base** command cannot be abbreviated. They must be whole words.

If the file transferred is a data file with information in either the "SDF" or "DELIMITED" format, you must first CREATE a database with a structure *identical* to the one from which it was sent. When finished, enter one of these two sequences:

SDF:
USE {Name of **H & D Base** File}
APPEND ALL FROM {Name of SDF File} SDF

DELIMITED:
USE {Name of **H & D Base** File}
APPEND ALL FROM {Name of DELIMITED File} DELIMITED

The information should now reside in the **H & D Base** data file.

Sending Files

In order to transfer a **H & D Base** data file to another database program, first format it in "SDF" or "DELIMITED" format using the "COPY" command. These are the correct forms:

SDF:
USE {Name of **H & D Base** File}
COPY ALL TO {Name of SDF File} SDF

Note: You may copy specific fields and records using the COPY command as outlined in the "Reference" section.

DELIMITED:

```
  USE {Name of H & D Base File}
  COPY ALL TO {Name of DELIMITED File} DELIMITED
```

Note: You may copy specific fields and records using the COPY command as outlined in the "Reference" section. You may also specify an alternate delimiter. If an alternate delimiter is *not* specified, double quotes will be used.

If the file is to be used in another Atari ST database program, just load that program and proceed. If the file is to be used with a database program on another computer, it must be sent to that computer via a modem or direct connect cable (you must have communications software packages on both machines to do so).

All COMMAND FILES generated by **H & D Base** are in "SDF" format and should be able to be run by the "dBASE II" program on other computers. If problems arise, check to make sure that no "tildes" (~) appear at the end of the first line in continuous line commands. In "dBASE II" they must be semi-colons.

Appendix C
H & D Base File Structure

DATA FILE HEADER

BYTE	CONTENTS	DESCRIPTION
0-3	4 Bytes	Date (YYMMDD) - Right Justified
4-7	32 Bit Number	Number of Records
8-9	16 Bit Number	Number of Fields
10-11	16 Bit Number	Record Size
12-31	20 Bytes	Reserved (Password)
32-51	20 Bytes	Field Descriptor #0
52-71	20 Bytes	Field Descriptor #1
"	"	"
"	"	"
1992-2011	20 Bytes	Field Descriptor #97

DATA FIELD DESCRIPTOR

BYTE	CONTENTS	DESCRIPTION
0-10	11 Bytes	Field Name - First Chr. = Count
11	1 Byte	Field Type = C,N, or L (0,1,2)
12-13	16 Bit Number	Field Data Offset to Beg. of Buffer
14	1 Byte	Field Length
15	1 Byte	Field Decimal Count
16-19	4 Bytes	Reserved for Future Use

INDEX HEADER & BAT (2k)

0-1	02 Index Code	
2-3	Key Length (+ 1 if odd)	
4-5	Number of Blocks (BAT)	
6-105	Key Expression (str)	
106-109	(Reserved)	
110-111	1st Block # in Index	Block Allocation Table
112-113	2nd Block # in Index	
114-115	3rd Block # in Index	
"	"	
"	"	
2028-2029	960th Block # in Index	

H & D Base File Structure, cont.

INDEX BLOCKS (4k)

(Immediately following the INDEX HEADER & BAT)

CONTENTS (4 BYTES)	CONTENTS LENGTH = KEY (+1 IF ODD)
Number of Keys In This Block	(Blank)
Record #	Key Expression in ASCII (with Count)
Record #	Key Expression in ASCII (with Count)
"	"
"	"
Last Record #	Key Expression in ASCII (with Count)

Index

Index, cont.

Index, cont.

Index, cont.

Index, cont.

Holmes & Duckworth

micronomists

H & D Base

PROGRAM HIGHLIGHTS:

- Comprehensive Manual: Introduction, Starting-up, Tutorial, Reference, Appendices, & Index Sections Plus Handy Command Card
- On-Screen "Help" System
- Elementary Text Editor
- Powerful Report Generator with Saved Reports
- Sort on Any Field to Any Level
- Compatible with "dBASE II"® Command Files
- Compatible with all "SDF" and "Delimited" Data Files
- Not Copy Protected
- Over 275 FORTH Commands (Including "calls" to the GEM® Interface — *Atari "Developer's Kit" Required*)
- Includes Sample Mailing List Program (Written in ***H & D Base***)
- Maximum Records/File: Limited Only by Disk Capacity
- Maximum Fields/Record: 97
- Maximum Characters/Field: 250
- Maximum Characters/Record: 2000

H & D Base is a Relational Database Management System which allows novice and expert users alike to manipulate any type of data through the use of straight-forward, English-like commands. For the novice, ***H & D Base*** is an interactive data storage and retrieval package suitable for maintaining phone directories, club rosters, genealogies and more. For experienced users, it is a system development package containing its own programming language. A skilled programmer can use ***H & D Base*** to create a system for handling inventories, accounts payable and receivable, client lists, and other programs which would normally be written in more cumbersome languages. Regardless of the level of user expertise, ***H & D Base*** is *the perfect tool* for data management.

Atari 520ST®
Minimum One Drive
(Single/Double Density or Hard Disk)
80 or 132 Column Printer

Written by Chester Holmes and Oliver Duckworth for:

MIRAGE

4055 W. Shaw #108 • Fresno, CA 93711 • (209) 227-8369

® "Atari 520ST" is a registered trademark of Atari Corp. ® "dBASE II" is a registered trademark of Ashton-Tate, Inc.
® "GEM" is a registered trademark of Digital Research, Inc.