DATABASE MANAGER IN MICROSOFT® BASIC

An easy-to-use data management system that costs only pennies per program!

BY GREG GREENE

DATABASE MANAGER IN MICROSOFT® BASIC

BY GREG GREENE

TAB BOOKS Inc.
BLUE RIDGE SUMMIT, PA. 17214

FIRST EDITION

SECOND PRINTING

Printed in the United States of America

Library of Congress Cataloging in Publication Data

Greene, Greg.
Database manager in microsoft basic.

Includes index.
1. Data base management. 2. Basic (Computer program language) I. Title.
QA76.9.D3G726 1983 001.64 83-4877
ISBN 0-8306-0167-8
ISBN 0-8306-0567-3 (pbk.)

This book is dedicated to my mother, who taught me that nothing is impossible to achieve if you are willing to do your best, and my father, who showed me how.

Introduction

The age of the microcomputer is upon us. Many people have rushed out and bought a computer expecting that it would be a useful tool. Too often they have found that, while the machines themselves are capable of great things, the software or programs that enable them to function do not work. After purchasing their computers, they find that the knowledge required to program the machines is not as easy to acquire as they would like.

Like many of you, I bought my computer with the idea of using it for a number of tasks. "Sure," I told my wife, "It will keep track of all kinds of things, like recipes, Christmas card mailing lists, and the like." The trouble was that, although the machine could indeed do all those things, it would require a program for each separate task!

In order to bring some order to this seemingly never-ending process of writing innumerable programs for similar applications, I wrote the programs contained in this book. With these programs and others that you will be able to develop from them, you will be able to have your computer keep track of all those things that you want it to. You can use the programs if your computer system runs a disk-based version of Microsoft BASIC. This includes the Heath/Zenith line or any other machine that runs CP/M. The Radio Shack TRS-80 Models I and III both use versions of Microsoft BASIC, as do the Apple and many others. The programs are written in Version 5.1 of Microsoft BASIC, and were developed on a Heath H-8 using dual disk drives (100K per drive), and an H-19 terminal. Examples of the changes required to make the programs run on the TRS-80 and other machines are included.

If you have a microcomputer system that uses a form of Microsoft BASIC and includes a disk drive, and are interested in getting greater performance from it now, this book is for you. Even if you are just interested in seeing how a disk-based database for a microcomputer system can be implemented, this book is for you. The programs are presented in both source code and flowcharts so that you will be able to follow the development process. Thus, if the source is not directly compatible with your system, you can make the adjustments required.

Contents

The Equipment

This chapter takes a look at what a typical computer system is, how it is organized, and how it performs. The systems that are covered are those computer systems that have one or more disk drives. Disk drives are devices that are used by the computer for long-term storage of information. If your system does not have a disk drive attached to it, and if you are not planning to add one to it, you will not be able to use the programs and examples in this book.

THE COMPUTER SYSTEM

Figure 1-1 is a diagram of a typical microcomputer system, showing the major parts. In it the console is connected to a monitor, a printer, and a set of disk drives.

Computers are very, very dumb machines. They can do nothing without being told what to do, when to do it, what information to use, the information, and where to store it. They are also very, very fast machines. This is their advantage. Once they are instructed how to accomplish a specific task, they will do so in an incredibly short space of time. It is this speed of operation that results in their widespread use.

The computers that are widely used by the general public are computers that essentially do one of two things. Either they perform a dedicated task such as playing the nth variation of space villains or they store and process information. It is the machines in the latter category that we are interested in.

Information storage and processing is a task that computers can do well, if they are given the correct routines to work with. In the type of computer system that we will be dealing with, the processing of information takes place within the computer in the area called RAM. RAM can store both a program and data. (A *program* is a set of instructions that the computer will follow; *data* is the information it will use when it performs these tasks.) Different programs can use and manipulate the same data. Simply put, the program acts on the

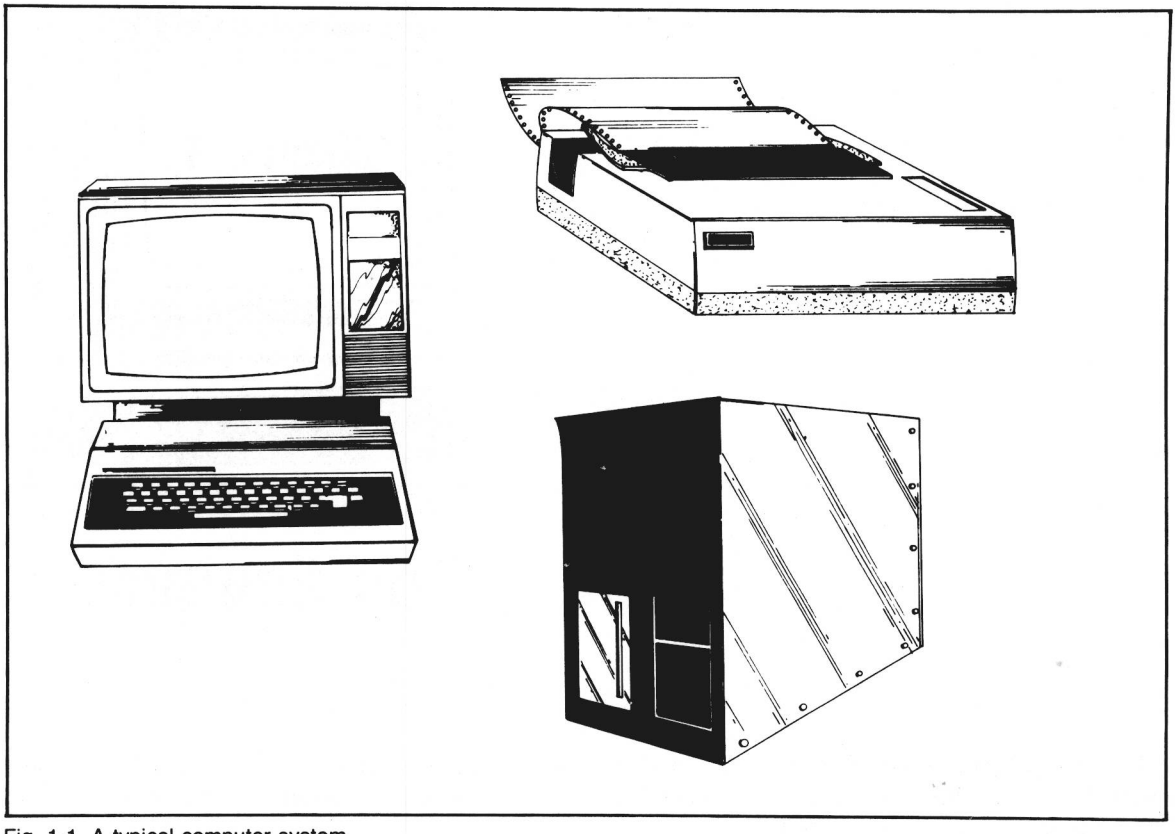


Fig. 1-1. A typical computer system.

data and may change it; the data is acted on by one or more programs but will never change the program. It is important to have this idea clear in your mind.

The data that the program works with is stored in one of two places in our system. Either it is on the disk, or it is in the computer's memory. If it is in the computer's memory, it can be changed and looked at very quickly; if it is on the disk then it cannot be changed or looked at quickly. Why then, you might ask, would anyone want the data on the disk, and not in memory? There are two reasons. First, the data in memory will be lost when the power is shut off. Second, the total amount of data we can keep in memory is limited. When the data is stored in memory, the limit is in the order of 64 thousand characters of information or less. These characters are usually referred to as *bytes*. A byte reflects the amount of memory required to store one character of information, and a *character* is typically a letter of the alphabet or a number.

Some machines, especially the newer ones that are coming out, can store a greater number of bytes in their memories, up to and even beyond 1 million bytes. These machines are not, as of the writing of this book, in great use. For the purposes of this book, we will concern ourselves with the computer system that has a limit of 64K of memory. (One K is one thousand bytes.)

Unfortunately not all 64K of memory is available for data storage. Some space is needed for the programs we will be using, and the computer needs some more space for the various routines that are required to get information to and from the terminal screen, printer, and disk drives and for other internal requirements. In point of fact, there is really

about 30K of usable memory space, and this will have to hold both the programs and the data. The disk drive, however, can hold a great deal more information. How much depends on your system. If you only have one disk drive, and it must hold some routines required by the system, you could, as in the case of a TRS-80 owner, be limited to about 30-40K per disk. If you have a system, like Heath or Zenith, that uses larger capacity drives, you can have up to 750K per disk. In any case, disk drives hold one great advantage over the 64K of memory in the machine: you can use as many of them as you need (or can afford). By using this method of multiple disks you can utilize your microcomputer for the storage and retrieval of information in a practical way.

Data and programs are stored on disks by a set of programs called an operating system. Sometimes the operating system is supplied by the manufacturer, and sometimes it is an extra that you must buy. If you have a Radio Shack TRS 80 computer, an Apple, or a PET the operating system that you have is one that is tailored to your particular system and is incompatible with most other systems. If you are using the operating system called CP/M, you have one that allows compatibility with any other system that uses CP/M, no matter what the make or model. If you are planning to use the data only on your own machine, all this is of little or no importance. If you are planning to use the information on different systems, it is of immense importance.

The actual writing of information to the disk and reading from it is done in terms of a unit of measurement called a *sector*. A sector is often 256 bytes long, although this varies according to the hardware you are using. Some systems using BASIC allow you to read in more or even less than one sector at a time. This will allow you to use every bit of space on the disk. Other systems, like Radio Shack, force you to read and write in single-sector units only. This may waste some space on the disk.

Look at the special diagram shown in Fig. 1-2. This is called a *memory map*. A memory map is a

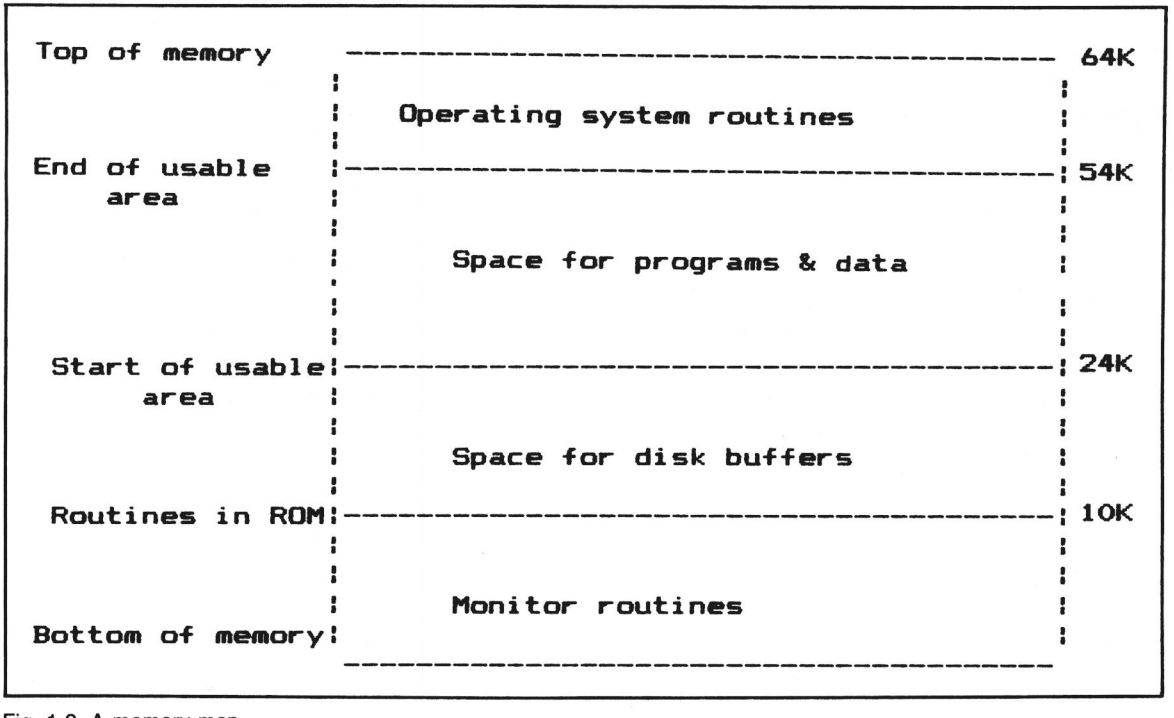


Fig. 1-2. A memory map

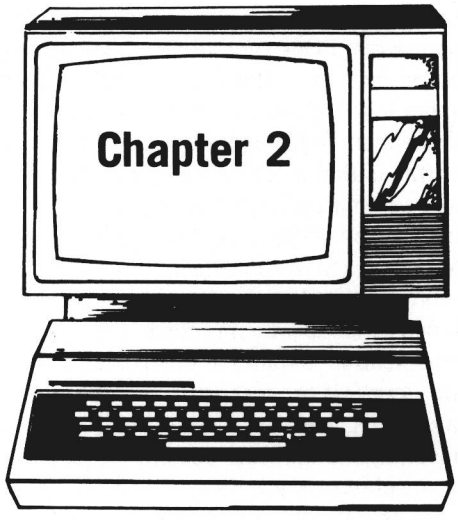
picture that tells you what parts of the computers memory are being used for what purposes. The map shown here is just an example, and although the one for the system that you are using will be different, the idea is the same. A memory map will tell you how much space is available in your system for you to store the program and the data in.

The numbers on the right side of the map show the approximate boundary locations for the various partitions. You would expect them to be different in your system.

One area of the map is labeled **Space for disk buffers.** Think of this space as a temporary area that is set aside for a single purpose. It is the area where information coming from the disk is placed so that the program can find it. It is also the place where information that is headed for the disk is put so that the operating system can find it. The size of this buffer area depends on the amount of space used by a sector on your system and the number of files that you have open at one time. In most systems you can have up to three files open, and the buffer area will be the standard sector size or 256 bytes. In some versions of Microsoft BASIC, the buffer area can be set to different sizes. This allows more information to be read from the disk and written to it at one time. This also allows the system to process several records at one time, but this is done at the cost of some memory space.

LANGUAGES

The language that you are using is a definite part of the overall system. Some languages let you do more than others do; some have special requirements or pose certain restrictions on the way that you will use the routines and examples in this book. The language you will probably use is a language called BASIC. In addition, it was probably written by the folks at MicroSoft. All the routines are in Microsoft BASIC release 5.2. For the most part, they can easily be changed to most other dialects of BASIC. Where we can, we will note the differences. If you are using Radio Shack equipment, you can use the routines almost entirely word for word, although there will be some differences in the screen formatting routines. If your particular version of BASIC is not covered here, don't despair; you can get a number of excellent books that will help you translate from one version to another. Drop down to your local computer store, or look through the listings of your favorite computer magazine to find the titles and prices that suit you best.



Introduction to the Database Manager System

This chapter examines just what a typical database is and why a database is desirable. Two types will be presented in general and the type dealt with in this book will be examined specifically. The kinds of features you might want to see in your own program will then be investigated. Next you will be told how to look at your system and determine which features you can implement. Then the organization of the information to go on the disk is explored. Finally, the code that will make up part of the foundation necessary for an effective program will be presented.

DATABASES

What is a database, and why do you need one anyway? A database is just a collection of data, any data. It can consist of anything that is important to the task at hand. A collection of the titles of the songs and selections in your record collection is a database. A collection of recipes is a database. In short any grouping of data is a database. Why do you need one? Chances are that you have a database already. Some people have small books full of the titles of music that they have. When they want to hear a particular piece, they first look up their selection in the book; see what record it's on, and then play that record. This works if they know the title of the piece and the book is indexed according to the titles. If they can only remember the name of the group, they have to check in another book that has the same information, but is indexed according to the names of the groups. The desired information could just as easily be a recipe, or the names and addresses of your clients and what kind of equipment that they have. The key point is that all the data is similar because the same type of information is being recorded about each person, song, or group; and you will want to access the information quickly.

Types of Databases

Having established the desirability of having a database on your system, what kind of databases are available? There are two kinds of databases, and

the difference lies in the way they organize the data on the disk. One type of database is organized in a hierarchal way. Each field of the data may have subfields that fall under the main field. Several of these main fields make up a record. The advantage of this is that a reference to a particular main field will automatically include the associated subfields. In this type of database it is difficult for the program to keep track of all the pointers that indicate which subfield is under which main field, and which main fields make up which record. This type of database organization is found in environments where there are very fast processors to look after all the pointers. What are the advantages of this type? One advantage is that it is possible to avoid duplication of any piece of data. If you have several records that have fields that contain the same information, like the name of a particular city, they will have a pointer in that field that points to the location of this piece of data. This can save an enormous amount of space in a large system, such as a government license registry. Space is saved because each record is only long enough to store data that is not duplicated in any other record. Thus the information in a given record may bear little relation to any other, but all information is organized in a strict hierarchy.

The second type of database is called a *relational database*. In this type, the records all have the same file structure. If a field contains the same information in each record, the disk will have the same information duplicated many times. The maintenance of such a system is extremely simple when compared to the hierarchal system. The program merely concerns itself with the contents of each field in a given record, without worrying about the other records. To find the contents of a field in this type of database organization, each record in the file must be accessed and the appropriate field checked. To find the contents of a file organized in an hierarchal file, the field concerned is checked and if the desired value is there, it will contain pointers that will indicate which records are currently pointing to this field and this field value. Those pointers can then be followed back to reconstruct the original records.

The hierarchal system, then, is better suited to the maintenance of a large database where rapid searching for a particular record, based on the information in one or more fields, is required. A relational database is better suited to the maintenance of smaller databases.

It is much easier to design and implement a relational database in a microcomputer environment, than it is to implement a hierarchal database. This doesn't mean it can't be done, because it has been done well; but the programs that do it are written in assembly language and are too difficult for the average computerist to do. This book will examine the relational database, since it will be much easier to develop and will serve you in looking after almost anything you will want to store.

Figure 2-1 shows a chart of how a relational database is organized. Each record will be in a row, and the columns will be made up by the fields in the record. For the purpose of this illustration, our database will be one which looks after five things; the name, address, city, zip code and state of each of the persons on a mailing list.

You can see in this chart that each record has the same organization as all the other records in the database. Each field stores its data in the same manner and contains data that is the same type in each record although the contents of each field may have a different value. In record 1 the value of the contents in the state field is MA, while in records 2, 3 and 4 the value is PA. The type of data is all the same, but the contents are different in one field. This illustrates some of the points of a relational system; the program can count on the data being of the same type and being in the same place in each record. Thus the fields in each record have the same relationships to each other as the corresponding fields in the other records do. You can also see that it is quite possible for each record to contain the same data as any other record, and in the chart above most of them do. While this does waste some space, it makes the maintenance of such a database very easy. For instance, in order to delete a field, you merely have to erase its contents, and to change data, you only have to modify the contents of one or more fields. You will find this easier than

Mailing List Database

Field:→	Name	Address	City	Z.C.	State
Record 1	Jones	123 Any	Bost	021	MA
Record 2	Mudd	234 Here	Phil	171	PA
Record 3	Lee	89 West	Phil	171	PA
Record 4	Lee	23 East	Phil	171	PA

Fig. 2-1. The organization of a relational database.

trying to develop a program that will keep track of the many pointers that would be necessary in a hierarchal database.

FEATURES OF THE DATABASE MANAGER SYSTEM

Now that the type of database has been established the features to be implemented in the system will be examined. Some basic decisions that will affect how the program is written will be made. Since the success of this program will depend to a large degree on how well you can adapt the code to your system, make sure you go over the next section of the book carefully. Prepare a list like the one we have shown, and make lots of notes!

Examining Your System

First, you must look at how your particular system stores data on the disk. You will recall the earlier discussion of how the operating system stores data on a disk. You must now determine if your system allows the use of random access disk operations. This is by far the norm. To find out, consult the manuals that came with your system. In all likelihood they will describe two methods of operation. First there is the sequential mode. In this mode, data is placed on the disk as it becomes available. If you have 50 records, the first one is written, then the second; then the third, until you have written the 50th record. This requires that if the information you want is on the 45th record, you must read the preceding 44 records first. This is not a practical way of doing things. Random access is much more efficient. The system should be able to look at any of the records on the disk by having the program state which record it wants to look at. If, in this mode of operation, the program instructs the computer to read record 45, it will do so without looking at any of the other records. This is not to say that you can't build a database system that uses sequential accessing, because you can. It's simply not a practical way of doing things. The time involved will make the operations too slow, and the increased disk wear will give you maintenance headaches.

Assuming that you are allowed by the makers of your system to utilize random access, the next thing to determine is whether or not you can have variable length records. A variable length record is a record whose length is adjusted so that it uses only the space required by the amount of data to be stored. If your recipe records only take up 125 bytes of space, you will only use that much per record. Some systems will allow you to use as little as two bytes in a record and as much as 4096 bytes. Most systems will require that you use 255 byte records, so the Database Manager uses this type of disk access. This means that you will waste some

space, but in the end, that could be an advantage. Using this procedure will simplify operations since the calling procedures for variable-length random-access records may be confusing.

Now take a look at what you want your system to accomplish. First, it must be able to write information onto the disk, read it, and provide some way to change and delete it. It must also provide a way to search for specific information that may be on the disk. It should have a way of reporting the information back in a manner that you can easily change. It would also be important to be able to sort the data to reflect current requirements for reporting. The system should also present you with choices for various operations via a menu. Most importantly, at the time that you create a file, you must be able to define just what information goes into it.

In order to see how much you can do with your machine, you must first take a look at what it offers. Get out the owner's manual for your computer and turn to the section that describes the features you can use. Write down the commands you have to use to accomplish the following procedures, if indeed you can do them at all.

Your Computer's Home Survey

1. Home the cursor and clear the screen.

   Your System:

2. Erase a line or part of a line.

   Your System:

3. Display characters in inverse video or half intensity.

   Your System:

4. Display characters on a status line, usually the 25th line of a display terminal.

   Your System:

5. Set up programmable special function keys if you have them.

   Your System:

6. Address the cursor to a particular spot on the screen.

   Your System:

Now, use your BASIC manual and find out how to do the following.

1. How to do error trapping so that you can recover from BASIC error conditions. These are sometimes unavoidable and unless you can prevent your program from crashing the program will be harder to convert.

   Sample statement: ON ERROR GOTO

   Your System:

2. How to use the intrinsic string functions, the most important of which is the ability to place a new group of characters within a string that already exists. You will also need to know about the LEFT$ and RIGHT$ functions.

   Sample statement: MID$(X$,4,7)=Y$

   Your System:

3. What the maximum length of your strings is. For most BASICs this is 255 bytes.

   Your System:

4. What fielding statements are required for random access files.

   Sample statement: Field #2,200 AS B$

   Your System:

5. How to direct print statements to the printer or the screen. For the most part it will be as simple as using LPRINT instead of the PRINT statement.

   Sample method: Poking the IO BYTE

   Your System:

6. How to open and close random files.

   Sample statement: OPEN "R", CLOSE #1

   Your System:

7. Whether or not you have a swap command that will let you sort easily.

   Sample statement: SWAP A$(X),A$(X+1)

   Your System:

8. Whether the index variable is checked and incremented at the beginning or end of a for-next loop. This will determine the final value of the variable when used as a counter.

   Your System:

9. How to access a character input at the keyboard without the return or enter key having to be pressed and without echoing the character to the screen.

   Sample statements: INKEY$,CIN,PIN,INPUT$(X)

   Your System:

Most of today's computers and terminals provide a way to tell the cursor where to go. This allows you to set up the program to present information in a pleasing manner. Being able to input information at the same spot on the screen is a great advantage. Some machines also allow you to highlight certain portions by using a reverse video technique. This turns on a block of text where the letters are black on white instead of white on black. Other features may also be available to you, such as being able to use special function keys that take on certain functions depending on what part of the program you are in. Because there are a great number of different protocols and ways of doing things, this book must have a standard system. In this standard system I will assume that you can tell the cursor where you want it to go. Other than that I will assume you have an ordinary terminal display. The terminal I use is a Heathkit H-19, so the protocols that I will be illustrating are the ones required for that terminal. They can easily be adapted for use on other equipment, and I'll show you how.

So what we have arrived at is that we want a database system that will do the following:

The Database Management System that will be presented in this book will do the following:

1. work with your disk drives
2. use 255 byte records
3. allow you to create different datafiles
4. allow you to add, change, delete, and modify records
5. allow you to search the database and find record(s) based on the contents of one or more fields
6. allow you to sort the records you have found based on the contents of one or two fields
7. allow the use of an addressable cursor

DATA ORGANIZATION

To organize your data you need to know how the data is to be organized on the disk. You need to know the kind of files that will be used and what kind of information will be stored there. You need to know how many records you can put on a disk, and how to organize them so that you can have different data in different databases that are all used by the same program.

If your system offers you the use of variable length records, you can set up a mechanism in the program that will count how much space each field takes and add these values together to get the total record length. You can then use this information to establish the record length on the disk. Since this is a relational database, if you accurately define one record in the database, you have done so for the rest. This technique will save space on the disk and allow you to have more records per disk. How many records? That depends on the size of your disk. If you are using a dual disk system, you can store the data on one disk, and the programs on the other. In the Heath system that this series of programs was developed on, I can use 90K for program storage on an initialized disk. If you have records that are 200 bytes long and you can use variable length records, you can store 90,000/200, or 450 records.

Using variable length records has a major disadvantage: you cannot add a field later because all the space is taken up. There are a number of ways to avoid this. You can set up an empty field, if you can foresee the inclusion of more data in a record at a

later date. You can use a program to bring data from one database to another, and by setting up the new field in the new database, transfer the fields you wish. You can use 255 byte records, and redefine the database by duplicating the specifications that you already have, and then adding an extra field at the end. If you have not exceeded the 255 byte limit and have not changed any parameters of the original fields, you are away to the races!

Now that you have some idea of the format of your data files, it must be determined how to set them up so that you can use the same programs to create and maintain different data in different databases? The answer is to create a separate parameters file that will act as a sort of dictionary. Whenever the program needs to look at a field for any reason, it will consult the dictionary and from it, the program will be able to tell where the field is in the record structure, what kind of data it contains, how long it is, whether or not there is a range limit imposed on it by the user, and whether or not it is stored in a compressed mode. Since this dictionary file is likely to be small in comparison to the database, you can store it in a sequential file on the disk that contains the program. This will leave more space on the data disk for our datafile and give the program access to the dictionaries of more than one database. Also we can write other programs that need only consult this dictionary to find out everything they need to know about a particular database in order to perform specific tasks, such as label making. In this way we add to the flexibility of our system.

The information concerning each record must be transferred to and from the disk. Most BASICs that allow random access allow the user to specify a certain number of fields in each record. Each field is assigned a variable name, and when a particular record is accessed, the information within the record is assigned to the variables. The way that this is done is as follows. First BASIC sets aside a 255 byte area in memory called a disk buffer. The field statements assigns portions of this area to certain variables. To do this each variable is assigned a pointer that tells it where its information begins in the buffer. BASIC will assign all the information from the start of that area character by character, until the start of the next variable is reached. When the information is retrieved by the get command, the assigning of the information is done automatically. When the put command tells the program to place the information back to the disk, all the information that is currently stored in the field variables, will be transferred to the buffer and thence to the disk. In order to protect the variables from getting lost, it is important that they not be used outside of a field statement or a LSET or RSET statement. Figure 2-2 shows the fields graphically.

You will encounter a problem if you use this method. You don't know at the start of the program how long the fields are to be because they haven't been created yet. You could get that information from the data dictionary, but you would have to substitute it into the field statements and that leads to a lot of code. There is a much easier way. Since any string variable in BASIC can be 255 bytes long, why not simply field a single variable, and then use the string functions of MID$, RIGHT$, and LEFT$ to take the string apart and find the information that we want? This would work provided we copied the information from the field string into a temporary string prior to use and put any new information back into the field string prior to putting it back on the disk. This technique works for both those systems that use variable length random-access records and those that use fixed length ones. In order for this to work well, you must depend on the data dictionary to indicate where each field starts in the string, how long it is, and whether the data is numeric or alphanumeric. Since you can use three types of numeric data: integer, single precision, or double precision, you can take advantage of the fact that they can be stored in 2,4 or 8 bytes respectively by converting them into strings. Thus you can save space on the disk. You will need routines to separate the information from the string and also to repack it. A graphic representation is shown in Fig. 2-3.

The data dictionary will have to store several pieces of information about each of the data fields in our data base. In order to place this information onto the disk and to recall it easily, it will be stored