SAMS

22397

ATARI® Trivia Data Base

Date Base and Trivia Game

James F. Hunter
Troy Rondot

LIMITED SOFTWARE WARRANTY

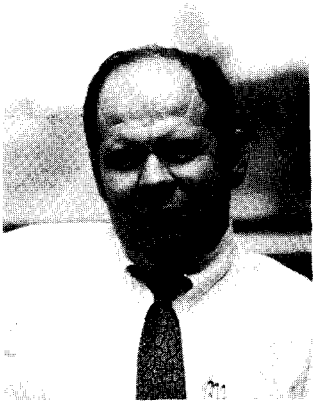
This warranty applies only to the software portion of this product. If you purchased this book by itself, without the companion diskette or cassette tape, then this section does not apply to you.

For a period of ninety (90) days from the date of original purchase at retail, the warrantor, identified below, warrants this software to load and run as a basic program for the indicated microcomputer model, to be free from defects in material and workmanship and to be merchantable and suitable for its stated purpose for the period of this warranty. This warranty may not be enlarged except in writing, signed by warrantor. THE WARRANTOR EXPRESSLY DISCLAIMS ANY IMPLIED WARRANTY INCLUDING THE WARRANTY OF MERCHANTABILITY AND THE WARRANTY THAT THE SOFTWARE IS SUITABLE FOR ITS STATED PURPOSE AS OF THE DATE NINETY (90) DAYS FROM THE ORIGINAL PURCHASE OF THE SOFTWARE AT RETAIL.

In the event of defect, malfunction or failure of the software to conform with this warranty, the warrantor will repair or replace the software at no cost to you. For warranty service, you should return the software to the warrantor, **Howard W. Sams & Co., Inc., Attn: Sams Software, 4300 W. 62nd Street, Indianapolis, Indiana 46268.** Software received damaged as a result of shipping will require you to file a claim with the carrier. This warranty gives you specific legal rights and you may also have some other rights which vary from state to state.

THIS WARRANTY IS LIMITED SOLELY TO THE ABOVE AND THIS WARRANTY AND ANY WARRANTIES IMPLIED BY STATE LAW WILL APPLY ONLY FOR THE PERIOD SET FORTH. (SOME STATES DO NOT ALLOW LIMITATION ON HOW LONG AN IMPLIED WARRANTY LASTS, SO THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.) THE WARRANTOR WILL NOT BE LIABLE FOR ANY LOSS, DAMAGE, INCIDENTAL OR CONSEQUENTIAL DAMAGES OF ANY KIND, WHETHER BASED UPON WARRANTY CONTRACT OR NEGLIGENCE, AND ARISING IN CONNECTION WITH THE SALE, USE OR REPAIR OF THE SOFTWARE. (SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.) UNLESS OTHERWISE CONTRARY TO STATE LAW GOVERNING THE PURCHASE, THE WARRANTOR'S LIABILITY SHALL NOT IN ANY CASE EXCEED THE CONTRACT PRICE FOR THE SOFTWARE CLAIMED TO BE DEFECTIVE OR UNSUITABLE.

Atari® Trivia Data Base



James F. Hunter is currently Director of Publishing for Howard W. Sams & Co., Inc. (ITT). A graduate of the University of California (Riverside), Jim is a veteran of seven years' experience in the personal computer field. In his spare time, he plays all board games with an enthusiasm and facility that sometimes astonish his opponents.

Troy Rondot, a 1983 graduate of Indiana University with a B.S. in Quantitative Business Analysis, is a computer consultant. He has worked with micros for the past six years. In addition to being a computer "hacker," Troy enjoys camping and hiking with his wife and daughter.

Atari® Trivia Data Base

by

James F. Hunter and Troy Rondot

Howard W. Sams & Co., Inc.
A Publishing Subsidiary of ITT
4300 West 62nd Street, Indianapolis, Indiana 46268 U.S.A.

A publishing subsidiary of ITT

FIRST EDITION
FIRST PRINTING—1984

International Standard Book Number: 0-672-22397-X
Library of Congress Catalog Card Number: 84-51541

Edited by *Susan Pink Bussiere*

Illustrated by *John E. Hopper*

Printed in the United States of America.

Preface

All computers, including micros, are designed to emulate processes of the human mind. It is, after all, *we* who have defined the tasks assigned to computers, with the goal of freeing ourselves from tedious and repetitious tasks that can be done more quickly and efficiently by an electronic device.

It is not unexpected that, after working with computers for a while, we begin to regard them as intelligent living entities. While not accurate technically, such an attitude can be of use in discussing what the programs in this book are designed to do.

We need not be intimidated by the speed with which a computer can do repetitious tasks and calculations. Remember that, while doing those calculations, the computer need not be concerned with satisfying superiors on the job, raising children, paying bills, or trying to achieve goals set for itself, by itself. In short, a computer is not distracted by the state of being human.

For its part, the computer does not enjoy some of the very positive attributes of a human. It cannot think or feel or play. We can. Playing a data retrieval game against a computer would be no fun. The computer would always win. It is our own lack of perfection at manipulating and retrieving data that has accounted for the tremendous success of games like *Trivial Pursuit*™. As we shall learn, the process of information storage and retrieval in a computer is exact and describable. Not so with us humans.

How many times have you heard someone say, "That reminds me of a story." Why? What is the mechanism that links one thought or event to another in the human mind? I don't have the answer, but I do believe that the lack of exact precision in describing those links can be a source of entertainment for people.

We are by nature curious. In the exercise of that curiosity we amass tremendous quantities of information, some of which might not be of immediate or even long-term use. In an effort to justify the acquisition and retention of such bits of knowledge, we name them trivia and proudly proclaim ourselves true foun-

---

Trivial Pursuit is a Horn Abbot game licensed by Horn Abbot Ltd., owner of the registered trademark Trivial Pursuit.

tains of useless information. In simpler terms, we try to express our belief that knowing things for their own sake is rewarding and fun.

The purposes of this book, and the included programs, are simple. The first purpose is to learn about the concept of a data base program on a computer, how it is developed, and how it works. The second and perhaps more important purpose is to take advantage of the given data base by using it as a pool from which to draw questions for the random inquiry trivia game program. Its third purpose is for you just to have fun. It is our hope that you will find both educational benefit and enjoyment in this book.

JAMES F. HUNTER

A note to the reader

The programs in this book were not written as applications software but as educational examples of what your personal computer can do. All of the programs have been tested and work on the machine configuration for which they were designed. The programs, or subroutines, are unprotected. This means that you can modify them to better understand how they work or to fit a different machine configuration.

What is a Combo Pack?

Atari Trivia Data Base is available in two formats—as a book and as a book and disk combination, called a Combo Pack. A Combo Pack is a step beyond your average technical book. While many books give you programming examples through printed listings (which we do here), Combo Packs also provide the listings recorded on magnetic media, either disk, cassette tape, or both. If you purchased the book only, you can type the programs listed.

Every effort has been made to be clear, concise, and informative. If you experience any difficulty with the software operations, the solution can be found in the book or in your computer manuals.

We are rather proud of the time and effort that went into preparing the Combo Pack. If you have purchased the Combo Pack and have enjoyed using it, let us know your thoughts. Your comments will be valuable in preparing future Combo Packs.

Loading instructions

If you bought this book as part of a Combo Pack, a disk is included. This disk contains the program listings printed in the book and a file of sample trivia questions. You must first make a working copy of the Combo Pack disk. To make a working copy, do the following:

To Make a Working Copy

1. Turn off the power to the computer and all peripherals. Turn on the disk drive.
2. When the red light goes out, insert the Atari DOS 2.0S disk into the drive and turn the door lever down.
3. When the red light goes out, turn on the computer. Type **DOS** and press <RETURN> to start the disk operating system.
4. Remove the DOS disk and insert a blank disk into the disk drive. (Label the blank disk "Working Copy" disk.) Close the door lever.
5. When the red light goes out, type **I** and press <RETURN> to format the Working Copy disk.
6. The screen displays *WHICH DRIVE TO FORMAT?* Type **1** and press <RETURN> for drive #1.
7. The message *TYPE "Y" TO FORMAT DISK 1* appears on the screen. Type **Y** and press <RETURN>
8. When the red light goes out, remove the Working Copy disk and place the Combo Pack disk into the drive. Close the door lever.
9. When the red light goes out, type **J** and press <RETURN> to duplicate the disk.
10. The computer displays *DUP DISK-SOURCE,DEST DRIVES?* Type **D1,D1** and press <RETURN>
11. The instruction *INSERT SOURCE DISK,TYPE RETURN* is shown on the screen. Press <RETURN>
12. After a few moments, the message *INSERT DESTINATION DISK,TYPE RETURN* is displayed. Remove the Combo

Pack disk and insert the Working Copy disk. Close the door lever.

13. When the red light goes out, press <RETURN>
14. After a few moments, the instruction *INSERT SOURCE DISK,TYPE RETURN* is shown. Remove the Working Copy disk and insert the Combo Pack disk. Close the door lever.
15. When the red light goes out, press <RETURN>
16. After a few moments, the screen displays *INSERT DESTINATION DISK,TYPE RETURN.* Remove the Combo Pack disk and insert the Working Copy disk. Close the door lever.
17. When the red light goes out, press <RETURN>
18. This completes the copying of the files on the Combo Pack disk to the Working Copy disk. Put the Combo Pack disk in a safe place.
19. Type **H** and press <RETURN> to tranfer the DOS files to your Working Copy disk.
20. The question *DRIVE TO WRITE DOS FILES TO?* appears on the screen. Type **1** and press <RETURN>
21. The message *TYPE "Y" TO WRITE DOS FILES TO DRIVE 1* is displayed. Type **Y** and press <RETURN>
22. When the computer is finished writing and the red light goes out, remove the Working Copy disk. You can now use the Working Copy disk to boot (start the program without using the DOS master disk).

To load the programs from the Working Copy disk, perform the following steps:

To Load the Programs

1. Insert the Working Copy disk into the drive and close the door lever.
2. When the red light goes out, type **B** and press <RETURN> to run Atari BASIC.

3. Type **RUN"D:DATABASE"** (you must type the quotes) to run the data base program.

OR

4. Type **RUN"D:GAME"** (you must type the quotes) to play the trivia game.

Contents

Chapter 7

Chapter 8

Chapter 9

Chapter 10

Chapter 11

Chapter 12

Appendix A

Appendix B

Appendix C

Chapter 1

Introduction

In the 1940's, my father tuned pipe organs as a sideline, and he later worked on the first electronic organs. In the process of repairing and tuning organs, he kept coming across the fact that the twelfth root of two is a very significant number in understanding the tempered musical scale. He needed to understand that scale thoroughly to do a good job of tuning. Because he worked principally as a motion picture projectionist at the time, he had ample opportunity to set about calculating by hand the twelfth root of two. At that time (and for some time prior) there were logarithms that could provide the information to six decimal places. However, intellectual curiosity and available time combined to motivate him to calculate the root by mechanical calculator and (after the calculator burned up) by hand.

The process was simple, if time consuming. He would pick a number between one and two, multiply it by itself twelve times, and see how close the result was to 2.00000. If the result of the multiplications (all done by hand) was greater than 2.00000, he reduced the trial number. If it was less than 2.00000, then he increased the trial number.

Over a period of months (and great perseverance), he was able to calculate the twelfth root of two to eight decimal places. Ironically, a few months after completing this exercise, my father discovered some log tables that rounded to ten decimal places, and was naturally able to redo the process in a period of minutes, as opposed to months.

In 1975, I purchased a Hewlett Packard calculator. With a few keystrokes I found the log of two, divided it by twelve, and took the antilog. In a matter of *seconds,* I'd obtained the same answer that it had originally taken my father *months* to arrive at. That speed in calculation is really what computers are all about. From the earliest modern computer, which was used by the U. S. Army to calculate mortar trajectories, to today's mainframes, minis, and microcomputers, the object has always been the same—to relegate repetitious and time-consuming tasks to electromechanical (and now silicon technology) devices.

A repeated calculation to obtain a mathematical answer is popularly called *number crunching*. It was not long after the advent of computers, however, when other kinds of activities, which had previously been done by hand, were being done by computer. One very obvious application is financial accounting, with its ledgers, balance sheets, T accounts, and profit and loss statements. One has but to remember the frustration of trying to balance a checkbook to understand the relief felt by accountants with the introduction of computerized bookkeeping.

As computers have become smaller, more affordable, and more powerful, other applications have been defined and implemented. Specifically, the applications which are of interest here are those currently being used on microcomputers, such as the Atari 800 XL. It is helpful to understand general groupings of those applications, so as to put the programs contained in this book and the accompanying disk in perspective. There are five main categories of microcomputer software, and each has several sub-categories:

Accounting
- General Ledger
- Accounts Payable
- Accounts Receivable
- Inventory
- Payroll

Productivity Tools
- Word Processor
- Spreadsheet
- Data Base
- Communications
- Graphics

Education
- Tutorial
- Skill Remediation
- Drill and Test
- Programmed Instruction
- Simulations

Entertainment
- Shoot-em-ups
- Strategy Games

- Fantasy Games
- Simulations

Utilities
- Programming Aids
- Communications
- Graphics

Looking quickly at the preceding list reveals that in this book we are dealing with an area of productivity tools called data bases. Before we begin construction of our data base, and subsequently use it for an amusing trivia game, we must describe and understand just what a data base is. And that's what is coming up in the next chapter.

Chapter 2
What is a Data Base?

The term **data base** in itself is quite descriptive. A collection of information arranged in some non-random and accessible order is a data base. Your telephone white pages are a data base. The phone book gives multiple iterations of the same types of information for many listees—last name, first name, address, and telephone number. *The Joy of Cooking* cookbook is a collection of recipes, each of which has ingredients and step-by-step directions for the preparation of particular dishes. It, too, is a data base. Given just these two examples, think about what other everyday sources of information could be regarded as data bases.

To better understand what electronic (computerized) data bases consist of and do, the following analogy to a commonly used manual system of maintaining a data base may be helpful. That system is the ever present 3 x 5 filing card system. Many such index card systems are the result of our desire to collect. Once our collection reaches a significant size, we need to have control of its contents. This control can be used to trade, to sell, to insure, to value, or for any number of other activities. For whatever reason, we definitely need to control the collection's contents.

Let's suppose that we collect cassette tapes of old-time radio show broadcasts, and we want to be able to share them with friends. Our collection has grown to over 600 shows, and memory alone will not suffice to summon up the exact details of each show. Our friends ask if we have any Jack Benny shows. We know there are two, but where? Time for an index file! Time, indeed, for a data base!

Simply writing down the information about a show, in paragraph form for instance, quickly proves of limited use. For example:

> The Jack Benny show broadcast June 4, 1938. Guest stars include Edgar Bergen and Charlie McCarthy. The show was sponsored by Lucky Strike, and is currently located in my upper left-hand desk drawer, on the Sony tape with the red and black label.

This card certainly has all the information we want, but after we have finished ten cards or so, we begin to file them. How? Alphabetically, of course. But alphabetically by what criteria? For starters, let's do it by the name of the show. What is the name of the show: "T" for the, "J" for Jack, or "B" for Benny? We obviously need some standard procedures. Also, when flipping through the cards, we need to be able to find the information on the show name quickly. Let's put the show name on a separate line at the top of the card.

The next thing we need to read on the card is the date of broadcast. Let's put it in the upper right-hand corner. And the guests . . . second line, left-hand side. We are quickly designing a format for the information. Ultimately, it could end up looking like this:

Show title:	Bdcst Date:
Guests:	Sponsor:
Location:	Running time:
Additional Comments:	

At this point, let's digress for just a moment to point out another phenomenon resulting from the increasing use of microcomputers. It has been dubbed *computerphobia,* and it is often seen in the following form: the media has convinced us all that we must be "computer literate" if we are to survive economically and socially in the next decade. We also are "required," if we would be thought "good parents," to provide "computer

literacy" for our children. Otherwise, perhaps we could be seen as impeding their intellectual advancement and success potential as they grow up and enter a world controlled by computers. However, we sometimes feel inadequate (if not plain stupid) because we don't understand microcomputers. If we admit it by asking questions, people will then learn the worst—we really *are* stupid.

Fortunately, this is not at all the case. Most often, it is not the *concept* that we do not understand, it is the *jargon* used to describe computers and their uses. An example may help to shed light on this point: following are two versions of a paragraph describing the use of our filing card system. The first uses terms with which we are all familiar, thus the description and concept are easily understood. The second, however, substitutes terms that would be used to describe the same data base if it were in a computer environment. By comparing the two, you see that (1) you can understand concepts of microcomputer usage, and (2) you are definitely *not* stupid.

Ordinary version

> Now that we have the information layout, we can begin to fill out cards for each specific show. We can then file them alphabetically by show title. After all of the cards have been filled out, we can use the newly formed card file. If, as time goes by, we change our collection, we can remove cards, change cards, or insert cards to reflect those changes. We can also decide to file them alphabetically by another bit of information on each card, such as Guest Stars, and re-sort them for location using that new category.

Computerese version

> Now that we have the format, we can begin to enter data for each specific show. We can then sort the data records by show title. After all of the records have been entered, we can access the newly formed data base. If, as time goes by, we change our collection, we can delete records, modify or edit records, or add records to reflect those changes. We can also decide to sort them alphabetically by another field, such as Guest Stars, and resave all records for access using the new key field.

In general terms, then, a computerized data base does the same things as a card system. So why bother with creating and maintaining a data base? Because a computerized system can do many other things that the card system can't; it can also do them far more quickly and easily. For example, let's suppose we wanted to find a show that was exactly 28 minutes long. Using the filing card system, we would have to check each card by hand or re-sort the cards by show length, from shortest to longest or vice versa. On the computer, we can search on the part of the card (field) that has the information until such a show is found, and then read (access) that whole record. Such sorts and searches, using a variety of keystrokes, are the backbone of an electronic data base.

Next, let us suppose we also had established a dollar value for each show, and entered that value in our format in its own location (field). A sophisticated data base would also allow us to add all of the values, thus giving us a total value for the collection at any point in time.

Finally, we shall assume that our insurance company wants limited information on each show in order to issue a policy on the collection. Using the filing card system, we would have to copy the cards. With an electronic data base, we could design a new format for printing the data, in columns for example, and summarize all of our shows on a few pages.

In summary, an electronic data base allows the creation of data entry and output formats, and the actual entry, storing, retrieval, editing, deletion, searching, and sorting of records.

In addition to these features, our trivia data base will be complemented by a random inquiry program. Our data in the trivia data base program will consist of questions and answers. The second program (the trivia game) will randomly select questions from the data entered into the data base program itself, compare our answers to the answers in the data base, and then score us on the speed and accuracy of our replies.

Chapter 3

Where Do We Begin?

Now that we understand something of what a generalized electronic data base is and what it does, we can begin to construct our own specialized random inquiry version of a data base. Beginning with Chapter 5, the method of this book is as follows: first, define the aspect of our data base program to be dealt with; and second, present and explain the BASIC language code that achieves the result. As a tool for following the logical flow of both programs, we use standard flowcharting techniques.

Such a process of program development is referred to as modular programming. Each task has its own section of code; and when we put them all together, we will have a program that meets our original design specifications.

Before we begin, however, let's review in more detail what we want our program to do. First, we want to be able to enter trivia data questions and answers (in pairs). Next, we want to be able to edit (add and alter) those entries until we are satisfied with the results. Finally, we want a second program that enables the computer to ask us those questions randomly, and give us individual and comparative scores for our efforts.

Perhaps a few comments on programming techniques and style would be helpful at this time. With regard to structure, the BASIC language can be something of a trap for the unwary. If we are not careful, the necessity to access subroutines in other parts of our program could result in the creation of "spaghetti code" (a term meaning program code written in such a haphazard fashion that it "wanders" up and down, and decreases program efficiency and legibility as it goes).

Top down, or structured, programming is much to be preferred. It means to start at the top (the beginning) of the program and work out the details in a logical, sequential manner. This requires that we have a very precise idea of how the logic of the program will work before we write a single line of program code.

The assignment of variable names within the program is also of great importance. You can prepare a logical scheme for these

variables by giving each variable a form that can be recalled easily. In other words, make the variable names mnemonic. For example, in the data base program the variable A$ stands for the answer; Q$ for the question. Mnemonics are a great help as we reuse variables throughout the program.

REMark statements within the program help remind us of the function of modules. They also aid other programmers, who might at some later date be working with the program, in understanding the logic we are using.

Finally, we cannot always assume that a user will follow our directions. We must therefore allow for circumstances in which the input of the user is entered at the wrong time or is entered in an invalid form. The process of accounting for such events is called "error trapping." This process is nothing more nor less than anticipating inappropriate actions by the program users, and preventing those incorrect actions from causing the program to fail, or "bomb" as we say in the trade.

We are now ready to build the skeleton of the program. Next stop, the flowchart.

One popular misconception about computers (although *you* don't buy it for a minute!) is that they can think. Computers cannot think. They can only be programmed to evaluate a specific situation according to exact guidelines, and then perform certain calculations based on those guidelines. An example may be of some help here.

Let's suppose that we want to create a program that receives as input from a user his sex, height, weight, and age. Then we want to program the computer to return a message as to whether the person who input the data is underweight, at a healthy weight (for him or her), or overweight. Before we can begin writing BASIC code, we must define the process by which the computer can output the appropriate response.

For a first pass, let's assume we have a chart of appropriate weights for adults. We can program the computer to retain that chart, and look up (based on the user input) an appropriate weight based on sex, height, and age. For now, let's just assume that a program exists that will accomplish this task. We now have, in the computer's memory area, the target weight and the actual weight of our suject. Now we come to the kind of logical branching activities computers can be programmed for, and which lead people to believe that computers can indeed think.

Actually, all that takes place are tests on the data, with selection of the correct message based on the results of those tests.

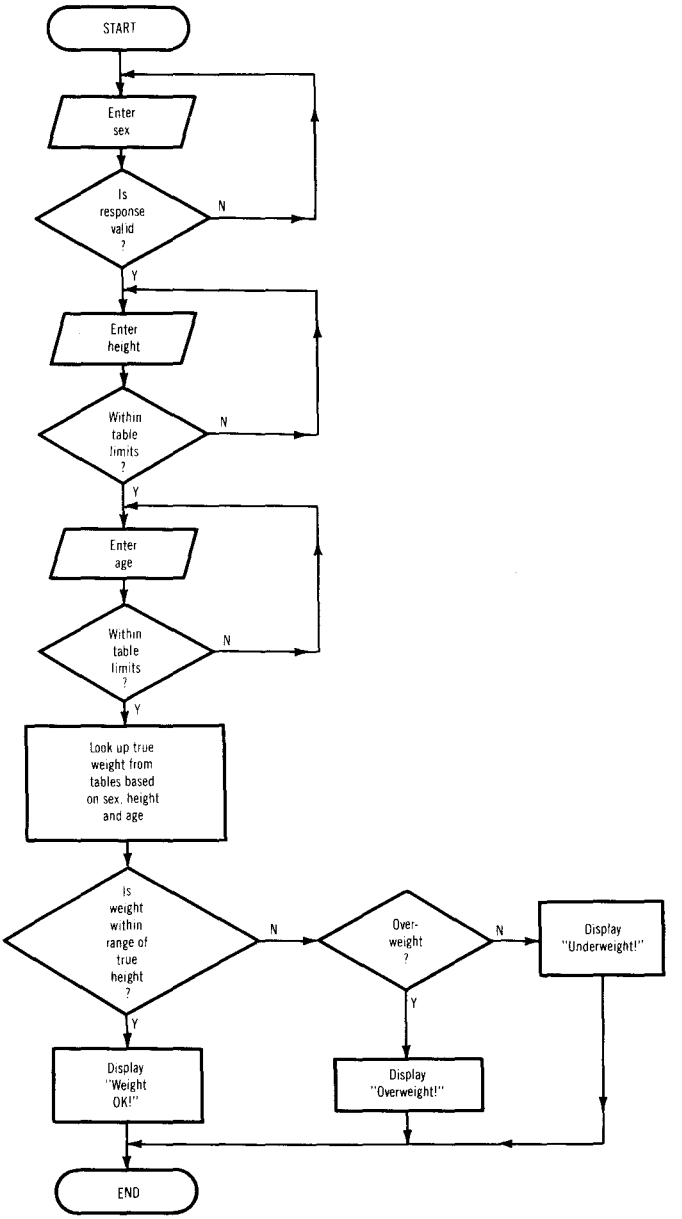


Fig. 3-1. Flowchart of the weight example program.

For the sake of this example, let us further assume that anyone who is plus or minus 5% of his best weight is close enough, and will receive a positive message. If his weight is less than 95% of his target weight, he will get a "skinny" message, and if he is over 105% of his best weight, he will read a "fat" message.

The preceding description of program flow is wordy, awkward, and not easily understood at a glance. There must be a better way, and there is. What we have here is a sequence of tests, decisions, and actions that can be represented nicely by a flowchart.

A flowchart is an outline of all the major steps necessary to perform a given task. Referring to Fig. 3-1, it is now clear that we receive input from a user, compare the actual and target weights, and then choose one of three messages to display on the screen. Even in this simple example, the value of the flowcharting tool is evident. In more complex programs (such as we are describing here), flowcharting is an invaluable aid to understanding what is happening in the program. Using Flowcharts 1-A through 1-I and Flowcharts 2-A through 2-E (which fold out of the back of this book), we will give an overview of the data base program and the trivia game program.

Chapter 4

Description of the Programs

Trivia Data Base

In the complete program listings (which appear in Appendices B and C) and variable listings (which are in Appendix A), you will note that this package actually consists of two different programs: a data base management program for file (question and answer) maintenance, and a game that utilizes those files. On the disk (included in the Combo Pack version), the laborious task of entering all of the code listed in the book has been eliminated, and both programs appear in final useable form.

Let's now look at the heart of our data base program (pull out Flowchart 1-A for simultaneous viewing). From the standpoint of the user (that's you), your first decision will be to load an existing data file, create a new one, or delete a file. If you choose to load an existing file or create a new one, you will be given three additional choices (decisions, decisions!). These lists of choices presented on the computer screen are called menus, and a program that always has menu options available on the screen is said to be menu driven (and sometimes even "user friendly").

The first menu that appears is the Main Menu. See Flowchart 1-C. Based on which option is selected, there are three different subprograms that are called into play. First, you can load an existing file, in order to add to, edit, or delete from the file in question. Next, you can create a new file, to handle questions about a different category, or replace an old group of questions. Finally, you can opt to delete a file, because after a while you will learn all of the answers, and the game wouldn't be fun anymore. These options are illustrated on Flowcharts 1-D, 1-E, and 1-F, respectively.

If you choose to delete a file, or load an existing file, the File Choice Menu is displayed on the screen. This menu lists the existing files from which you may choose. If you want to create a new file, you are prompted to enter the file name.

After you have chosen an existing file or created a new one, the Edit Menu is displayed. You again have three choices. On Flowchart 1-G, these options are to enter new data, edit old data,

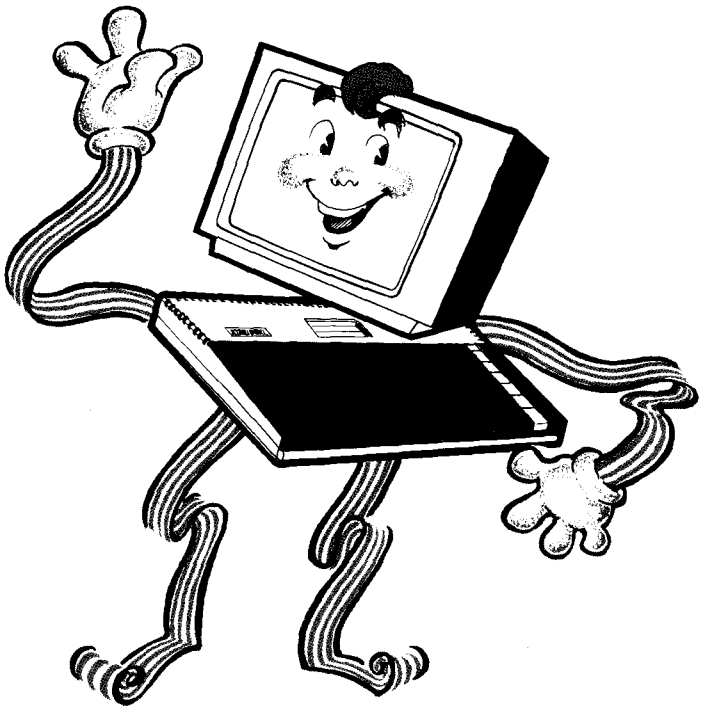


Fig. 4-1. Mr. User Friendly.

or return to the Main Menu. For the program flow for the first two options, see Flowcharts 1-H or 1-I, respectively.

If you choose option #1, enter data, the Question Entry screen appears. You can enter new questions and answers, which are simultaneously saved to disk, for use with the game program.

If you choose option #2, edit data, the Question Edit screen is displayed. You can choose the question and answer (the record) to edit by having the program search forward or backward in the file, or "jump" to a specified record number. After you have edited the question and/or the answer, the changes can be saved.

After all of this, you have a data base from which the game can randomly call forth questions, and score you and your friends (at least when the game starts, they're your friends) on your knowledge of the subject area.

Trivia Game

Aside from the instructional value of learning programming concepts through the development of both programs, the trivia data base and trivia game serve some other purposes.

First, everyone who has wanted to play catch with a football, but couldn't find a playing partner, will attest that such undertakings aren't much fun alone. With these programs, however, it is possible to practice (cheat?) and improve one's skills. Also, the record and time keeping chores in such a fast-paced undertaking as the trivia game is handled by the computer, leaving the contestants free to deal with more *trivial* matters.

So, let's look at how the game works. Please pull out Flowchart 2-A for simultaneous viewing. Pretty simple, isn't it! Variable initialization is done internally by the machine, then you choose a data file (which was created with the data base program), enter the number and names of players, and then you're off to the races.

In order to assure that none of the players (with a good memory) has an advantage, the order of the questions is scrambled. Fortunately, the corresponding answers are kept with the questions. Then a player is put on notice that a question is forthcoming. Ready! Set! Go! A timer starts, and the player must enter the correct answer. If the answer is correct, points will be awarded based on the elapsed time between question and answer. Obviously, if the answer is incorrect, no points are awarded. Each player will be asked five questions per game, and the highest total score wins.

A file containing 100 sample questions is included if you bought the Combo Pack. Not only has the recent interest in trivia resulted in several board games, there is even a magazine with oodles of new questions every month. Additional sources for data base material can come from schools, club activities, or any other aspect of life wherein rote learning is required.

That, then, is what this pair of programs is about. Chapters 5, 6, and 7 deal with an evaluation and discussion of the flowchart for the trivia data base program, and the resultant BASIC code that achieves that flow. Chapters 8 and 9 describe the code for the trivia game.

We hope that you find the program logic and code generation enlightening, and that you have as much fun using the program as we had in testing it.

Chapter 5

Main Menu

Let's now look at the data base program in detail. For descriptions of the variables, refer to Appendix A. Flowchart 1-A is an overview of the program. Chapter 7 explains all of the subroutines used in the data base program.

Initialization

The beginning of the program, the program initialization, sets up the key conversion routine, and initializes the variables and error trapping. See Flowchart 1-B. The key conversion routine is made necessary by internal protocols of the Atari. The Atari also requires that space be dimensioned (reserved) for all string variables used in the program. Error trapping is required in order to prevent unforseen errors, such as a "disk reading" error.

```
100 DIM KEY1$(64),KEY2$(64)
110 KEY1$="LJ;  K+*0 PU I-=V C  BXZ4 36 521, .
N M/ R EY TWQ9 07 8<>FHD  GSA"
120 KEY2$="LJ:  K\^0 PU I_!V C  BXZ$ #& % ![ ]
N M? R EY TWQ( )' @  FHD  GSA"
130 KS=PEEK(141)*256+PEEK(140)
140 POKE KS+94,34
150 DIM DIR$(20),Q$(54),A$(25),B$(54),F$(8),S(
100),B(100),N$(3)
160 FOR L=1 TO 54:B$(L,L)=" ":NEXT L:Q$=B$:A$=
B$(1,25)
170 GRAPHICS 0:TRAP 7010
180 GOSUB 6110
190 POKE 16,64:POKE 53774,64
```

Line 100 reserves 128 spaces in memory by dimensioning two strings (KEY1$ and KEY2$, which each contain 64 characters). Lines 110 and 120 fill these spaces with characters in accordance to the following rules:

- The position of all the characters is determined by their Atari keyboard scan codes. Each character's position represents its keypress scan code plus one. The scan code for an

uppercase character is 64 plus the scan code for its lowercase counterpart. (A cursory perusal of an ATASCII chart will show why this is true.)

- All lowercase letters are converted to uppercase because only uppercase letters are used while the program is running. This facilitates reading of the key scan codes.
- Spaces are used for special keys, such as <TAB>, <CLEAR>, and <RETURN> because these keys cannot be represented by a single character.

Perhaps some examples will make this more understandable. The scan code for a lowercase letter "l" is 0, thus, an uppercase letter "L" is placed in KEY1$ in position 1. (Position 1 is the value of its scan code plus 1.) The scan code for <RETURN> is 12, thus a space is placed in position 13. And finally, the scan code for an uppercase "T" is 109, thus an uppercase letter "T" is placed in position 110 (in KEY2$).

Because KEY1$ and KEY2$ are string variables, when lines 110-120 are executed ATASCII characters are placed in memory exactly as shown in KEY1$ and KEY2$. If you are typing the listings into the computer, you must type KEY1$ and KEY2$ exactly as shown (including the spaces).

Line 130 loads the starting location of these dimensioned characters into variable KS (the memory location where ATASCII characters are stored). This is done because the memory location of an ATASCII character can be derived by adding its scan code (K) to KS and PEEKing into memory. This operation is necessary because the scan codes do not follow in logical order. If all of this sounds confusing, it's probably because it is. However, do not despair. This procedure is a peculiarity of the Atari, and has little to do with the actual logic of the programs we're discussing. If you choose not to master the process of key conversion, just chalk it up to *Magic,* and let's move on.

Line 140 puts the ATASCII representation for a quote mark into the reserved area. The quote mark is used as a beginning and end delimiter (mark) for KEY1$ and KEY2$. If you placed a quote in either of these strings in the same way that the characters are placed in the strings, the quote would indicate to the program that it had reached the end of the string. Therefore, the quote must be placed into the reserved area using a POKE.

Lines 150 and 160 initialize the string and array variables that are used by the program. The string variables must be initialized to blanks because this is not an automatic feature of the Atari.

Line 170 sets the screen defaults to mode 0 and turns on the error trapping routine at line 7010 (see Chapter 7). Line 180 calls the file counting subroutine at line 6110, which counts the number of data files on the disk. Finally, line 190 disables the break key. This ensures that the program can't be stopped accidentally.

Displaying the Main Menu

When the program begins, the Main Menu is displayed (see Fig. 5-1 and Flowchart 1-C).

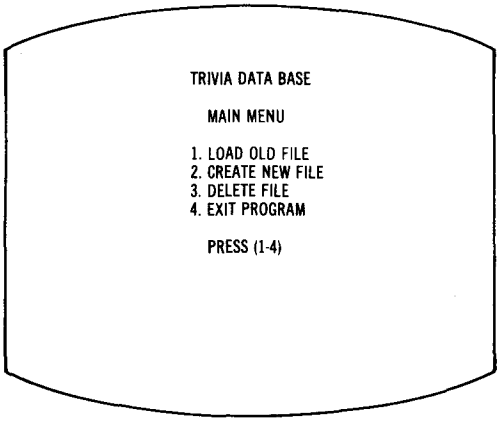


Fig. 5-1. Main Menu.

The code for this section is:

```
200 REM *** MAIN MENU
210 POKE 752,1:PRINT "{"
220 POSITION 11,2:PRINT "TRIVIA DATA BASE"
230 POSITION 14,5:PRINT "MAIN MENU"
240 POSITION 11,8:PRINT "1. LOAD OLD FILE"
250 POSITION 11,10:PRINT "2. CREATE NEW FILE"
260 POSITION 11,12:PRINT "3. DELETE FILE"
270 POSITION 11,14:PRINT "4. EXIT PROGRAM"
280 POSITION 14,17:PRINT "PRESS (1-4)"
290 GOSUB 6010:IF K2<49 OR K2>52 THEN 290
300 ON K2-48 GOTO 1010,2010,5010,8010
```

Line 210 turns off the cursor and clears the screen. Lines 220-280 print the menu on the screen.

Line 290 calls the key input routine at line 6010, and checks to make sure the keypress (K2) is valid (i.e., less than 49 or greater than 52, which is an ATASCII representation of a key input of 1 through 4). If the keypress isn't valid, the program returns to the key input routine to get another keypress.

If the user's input is valid, line 300 directs the program flow to the appropriate section. If option #1 (load an old data file) is chosen the program continues at line 1010. If option #2 (create a new data file) is chosen the program continues at line 2010. If either option #3 (delete a file) or option #4 (exit the program) is chosen, the program continues at lines 5010 or 8010.

Loading an old file

The code for choice #1, load an existing file, is as follows:

```
1000 REM *** FILE CHOICE MENU
1010 PRINT "{CLEAR}":POSITION 12,3:PRINT "FILE CHOIC
ES"
1015 IF F>9 THEN F=9
1020 IF F=0 THEN 1100
1030 OPEN #4,6,0,"D:*.TDB":FOR L=1 TO F:INPUT
#4,DIR$:POSITION 12,L+5:PRINT L;". ";DIR$(3,10
):NEXT L:CLOSE #4
1035 POSITION 12,F+7:PRINT "0. EXIT"
1040 POSITION 12,F+9:PRINT "PRESS (0-";F;")"
1050 GOSUB 6010:IF K2<48 OR K2>F+48 THEN 1050
1055 IF K2=48 THEN 210
1060 OPEN #5,6,0,"D:*.TDB"
1070 FOR L=1 TO K2-48:INPUT #5,DIR$:NEXT L:CLO
SE #5:F$=DIR$(3,11)
1080 IF F$(LEN(F$),LEN(F$))=" " THEN F$=F$(1,L
EN(F$)-1):GOTO 1080
1090 GOSUB 6710:OPEN #5,4,0,DIR$:INPUT #5;N$:C
LOSE #5:N=VAL(N$):GOTO 4010
1100 POSITION 7,6:PRINT "NO OLD DATA FILES ON
DISK":POSITION 12,8:PRINT "PRESS ANY KEY"
1110 GOSUB 6010:GOTO 210
```

See Flowchart 1-D. Lines 1010-1040 display the File Choice Menu (see Fig. 5-2). First, line 1010 clears the screen and displays "File Choices." Then line 1015 sets the maximum number of files to nine.

If there are no files on disk (F = 0), line 1020 transfers control to lines 1100-1110. This section displays a "no files" message,

waits for a user prompt to continue, and returns the program to the Main Menu.

If there are one or more files on disk, line 1030 opens the directory for file names with the extension "TDB" and uses a FOR...NEXT loop to display these file names. It then closes the directory. Lines 1035-1040 print the exit option on the screen and the message to press a key.

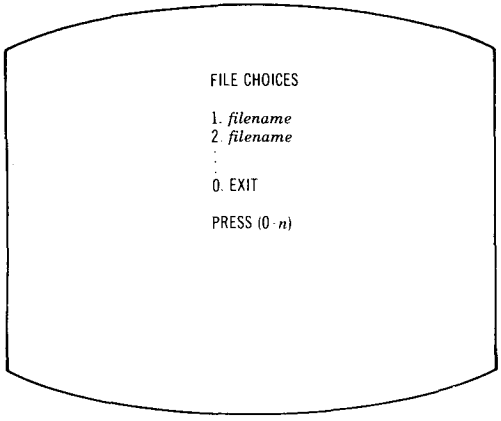


Fig. 5-2. File Choice Menu.

Line 1050 calls the key input routine at line 6010 to get your choice. It then checks the validity of the keypress. If your choice is invalid (i.e., not one of the choices displayed) the key input routine is again called. If you pressed 0, line 1055 returns the program to the Main Menu.

If your choice is valid and your input is not zero, the following occurs. Line 1060 re-opens the directory. Line 1070 reads in the file name you chose (F$), and line 1080 removes the blanks in the file name. Line 1090 calls the file name routine at line 6710. When the program returns from this routine, line 1090 inputs the number of questions in the file and then transfers control to the Edit Menu.

Creating a new file

The code for choice #2, create a new file, is:

```
2000 REM *** NEW FILE CHOICE
2010 PRINT "{CLR}":POKE 752,0:F$=""
```

cont. on next page

```
2015 IF F>=9 THEN 2090
2020 POSITION 7,5:PRINT "NEW FILE NAME ";:INPU
T F$
2023 IF F$="" THEN 2070
2025 FOR L=1 TO LEN(F$):IF ASC(F$(L,L))<65 OR
ASC(F$(L,L))>90 THEN 2070
2027 NEXT L
2030 POKE 752,1:TRAP 2060:OPEN #5,6,0,"D:*.*"
2040 INPUT #5,DIR$:IF DIR$(3,LEN(F$)+2)=F$ THE
N CLOSE #5:GOTO 2070
2050 GOTO 2040
2060 CLOSE #5:TRAP 7010:GOSUB 6710:OPEN #5,8,0
,DIR$:N=0:N$="0  ":PRINT #5;N$:CLOSE #5:F=F+1:
GOTO 4010
2070 POKE 752,1:POSITION 3,8:PRINT "ILLEGAL OR
 DUPLICATE FILE NAME":POSITION 11,11:PRINT "PR
ESS ANY KEY"
2080 GOSUB 6010:GOTO 210
2090 POKE 752,1:POSITION 8,8:PRINT "TOO MANY F
ILES ON DISK":POSITION 12,11:PRINT "PRESS ANY
KEY"
2100 GOSUB 6010:GOTO 210
```

Refer to Flowchart 1-E. First, line 2010 clears the screen and turns off the cursor. It then sets F$ (the file name) to null because we are creating a new file name. If there are nine or more files on disk (F>8), line 2015 transfers control to lines 2090-2100. This section displays a "too many files" message, waits for you to press a key to continue, and returns the program to the Main Menu.

If there are less than nine files on the disk (F<9), line 2020 accepts the user input of the new file name (F$). Line 2023 checks to see if F$ is blank. If so, control is transferred to lines 2070-2080, which print an "illegal or duplicate file name" message and the prompt to press a key to continue. Control is then returned to the Main Menu. Lines 2025-2027 check to see if the file name is valid (i.e., the file name contains only letters). If the file name is not valid, control is transferred to lines 2070-2080 (see above).

Line 2030 changes the location of where the program will continue if an error occurs from its original location (at 7010) to line 2060. (We will explain why in a moment.) It then opens the directory so it can be read.

Lines 2040-2050 read and compare each file name to F$ to search for duplicate file names. If there is a duplicate, control is

tranferred to 2070-2080 (see above). The program then goes back to read the next file name. When the program reaches the end of the last file, an error occurs. But because we have changed the location of the error trapping routine, the error trap at line 2060 gets program control. We have changed the location of the error trapping routine because we want the program to handle this error differently than other errors. It must be noted, because we have all been taught that "errors" are bad things, that in this context an error can be used constructively to assist in the direction of program control during execution. I had an Apple II customer who once was sure someone had broken his new computer, because "Syntax Error" appeared on the screen. Thank goodness you are much too sophisticated to have such a negative reaction to our programming friend, the Error.

Line 2060 closes the directory. It then calls the file name routine at 6710, which converts F$ to DIR$, and opens the data file (DIR$). Upon return from the file name routine, line 2060 sets N$ (the string representation of N) and N (number of questions) to zero, and prints N$ as the first record of the file. It then closes the file, adds one to the number of files (F), and transfers control to the Edit Menu.

Deleting a file

Choice #3, delete a file, is accomplished with the following code:

```
5000 REM *** FILE DELETION
5010 PRINT "{":POSITION 12,3:PRINT "FILE DELET
ION"
5020 IF F=0 THEN 5120
5030 OPEN #4,6,0,"D:*.TDB":FOR L=1 TO F:INPUT
#4,DIR$:POSITION 12,L+5:PRINT L;". ";DIR$(3,10
):NEXT L:CLOSE #4
5035 POSITION 12,F+7:PRINT "0. EXIT"
5040 POSITION 9,F+9:PRINT "DELETE WHICH FILE?"
:POSITION 12,F+10:PRINT "PRESS (0-";F;")"
5050 GOSUB 6010:IF K2<48 OR K2>F+48 THEN 5050
5055 IF K2=48 THEN 210
5060 OPEN #4,6,0,"D:*.TDB":FOR L=1 TO K2-48:IN
PUT #4,DIR$:NEXT L:CLOSE #4:F$=DIR$(3,11)
5070 IF F$(LEN(F$),LEN(F$))=" " THEN F$=F$(1,L
EN(F$)-1):GOTO 5070
```

cont. on next page

```
5080 PRINT "{CLEAR}":POSITION 12,10:PRINT "DELETE ";
F$;"?":POSITION 12,12:PRINT "PRESS Y OR N"
5090 GOSUB 6010:IF K2=78 THEN 210
5100 IF K2=89 THEN GOSUB 6710:XIO 33,#1,0,0,DI
R$:F=F-1:GOTO 210
5110 GOTO 5090
5120 POSITION 10,6:PRINT "NO FILES TO DELETE":
POSITION 12,8:PRINT "PRESS ANY KEY"
5130 GOSUB 6010:GOTO 210
```

See Flowchart 1-F. First, line 5010 clears the screen and displays the message "file deletion." If there are no files on disk (F = 0), line 5020 transfers control to lines 5120-5130. This section displays a "no files" message, waits for you to press a key to continue, and returns the program to the Main Menu. If there are one or more files on disk, line 5030 opens the directory for file names with the extension "TDB" and uses a FOR...NEXT loop to display these file names. It then closes the directory.

Lines 5035-5040 print the exit option on the screen and the prompt message to press a key.

Line 5050 calls the key input routine at line 6010 to get your choice. Then it checks the validity of the keypress. If your choice is not one of the options displayed, the key input routine is called again. If you pressed 0, line 5055 returns the program to the Main Menu.

If your choice is valid and your input is not zero, the following lines are executed. Line 5060 re-opens the directory, inputs the file name chosen (F$), and closes the file. Line 5070 removes the blanks in the file name. Line 5080 clears the screen and prints a confirmation message. Line 5090 calls the key input routine for a keypress; if the keypress is "N," control is transferred to the Main Menu. If the keypress is "Y," line 5100 deletes the file using an XIO command, subtracts one from the number of files, and transfers control to the Main Menu. If neither "Y" nor "N" was pressed, line 5110 returns to the key input routine.

Exiting the program

Choice #4 exits the program. Program flow is directed to statement 8010.

```
8000 REM *** EXIT
8010 PRINT "{CLEAR}":POKE 16,192:POKE 53774,247:POKE
 752,0:END
```

Line 8010 clears the screen and turns on the cursor. It also enables the break key (which was disabled to prevent someone from accidentally ending the program) and ends the program.

That's it for the explanation of the program initialization and Main Menu. By now you're probably used to simultaneously viewing the flowcharts and program notes. In the next chapter, we describe the Edit Menu and how it works.

Chapter 6

Edit Menu

There are three choices in the Edit Menu. You can enter new data or edit existing data in the data base, or return to the Main Menu. The entry and edit modules are undoubtedly the most complex in the data base program.

Displaying the Edit Menu

If you choose option #1 or #2 from the Main Menu (load an old file or create a new one), the next menu you see is the Edit Menu. See Fig. 6-1.

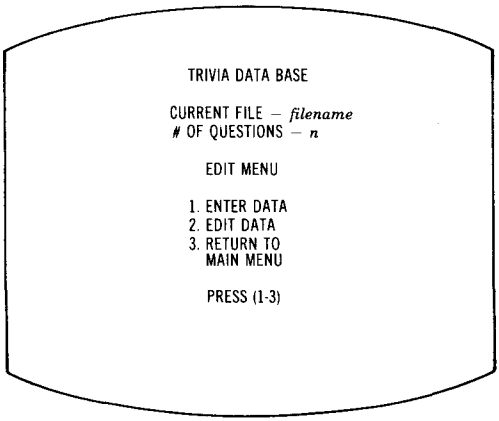


Fig. 6-1. Edit Menu.

The following code displays the Edit Menu:

```
4000 REM *** EDIT MENU
4010 POKE 752,1:PRINT "{CLEAR}"
4020 POSITION 11,2:PRINT "TRIVIA DATA BASE"
4030 POSITION 9,4:PRINT "CURRENT FILE - ";F$:P
OSITION 9,5:PRINT "# OF QUESTIONS - ";N
4040 POSITION 14,8:PRINT "EDIT MENU"
4050 POSITION 11,11:PRINT "1. ENTER DATA"
4060 POSITION 11,13:PRINT "2. EDIT DATA"
```

cont. on next page

```
4070 POSITION 11,15:PRINT "3. RETURN TO":POSIT
ION 14,16:PRINT "MAIN MENU"
4080 POSITION 13,19:PRINT "PRESS (1-3)"
4100 GOSUB 6010:IF K2<49 OR K2>51 THEN 4100
4110 ON K2-48 GOTO 3010,4120,210
```

Refer to Flowchart 1-G. Line 4010 turns off the cursor and clears the screen. Lines 4020-4080 display the current file name (F$), the number of questions in that file (N), and the Edit Menu options.

Line 4100 calls the key input routine at line 6010 (see Chapter 7) and checks to make sure that the keypress was a 1, 2, or 3. You have three choices. You can enter new data (#1), edit old data (#2), or return to the Main Menu (#3). Line 4110 directs the program flow to choices 1, 2, or 3, lines 3010, 4120, or 210, respectively.

Now let's look at the three choices on the Edit Menu in detail.

Entering data

If you choose option #1 from the Edit Menu, the following code is executed. Pull out Flowchart 1-H for simultaneous reference, refer to Appendix A for variable descriptions, and let's look at the logic for entering data. Remember, subroutines for this program are in Chapter 7.

```
3000 REM *** QUESTION ENTRY
3010 IF N>=100 THEN 3600
3020 GOSUB 6210:POSITION 13,3:PRINT "QUESTION
ENTRY":N=N+1:GOSUB 6410
3030 POSITION 9,21:PRINT "^R:RESTART    ^E:EXIT
";
3180 OPEN #5,9,0,DIR$:Q$=B$:A$=B$(1,25)
```

Line 3010 checks to see if the number of questions is greater than or equal to 100. If so, the program transfers to lines 3600-3610.

```
3600 PRINT "T":POKE 752,1:POSITION 14,8:PRINT
"FILE FULL":POSITION 12,11:PRINT "PRESS ANY KE
Y"
3610 GOSUB 6010:CLOSE #5:OPEN #5,12,0,DIR$:GOS
UB 6910:PRINT #5;N$:CLOSE #5:GOTO 4010
```

Line 3600 prints a "file full" message and prompts the user to continue. Line 3610 closes the data file and opens it at the beginning of the file. It then calls the number length routine at line 6910. Upon returning from this routine, line 3610 prints the number of questions in the file, closes the file, and transfers control to line 3010 for a different edit option.

Line 3020 calls the screen printing routine at 6210, which prints the question entry screen (see Fig. 6-2). Line 3020 then increments N (the number of questions) by one and transfers to the question number routine at line 6410, which prints the number of questions.

Line 3030 prints the entry control options (^R: Restart and ^E: Exit). Line 3180 opens the data file for input and output, and blanks the current question and answer.

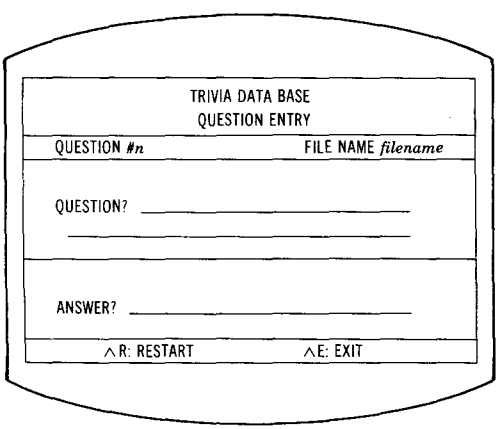


Fig. 6-2. Question Entry screen.

Entering first line of the question

As Fig. 6-2 illustrates, there are two lines on which to enter the question. Entering characters on the first line is handled somewhat differently than entering characters on the second line. The next section of code and program description deals with entering characters on the first line of the question. We will describe the code for entering characters on the second line afterwards.

```
3190 POKE 752,0:POSITION 13,9:PRINT " ";
3195 IF N>100 THEN N=100:GOTO 3600
3200 T=0:S=0
3210 R=3220:GOTO 3700
3220 IF K=12 AND FL=1 THEN 4610
3230 IF K=12 THEN 3450
3240 IF K=52 AND T>0 THEN POSITION T+13,9:PRIN
T " ←";:Q$(T,T)=" ":T=T-1:GOTO 3210
3270 IF K2=32 AND K<>33 THEN 3210
3280 T=T+1:Q$(T,T)=CHR$(K2):POSITION T+13,9:PR
INT CHR$(K2);
3300 IF T<22 THEN 3210
3310 IF K2=32 THEN POSITION 4,11:PRINT " ←";:G
OTO 3370
3320 FOR S=22 TO 1 STEP -1:IF Q$(S,S)=" " THEN
 3335
3330 NEXT S
3335 S=22-S:T=22+S:POKE 752,1
3340 FOR L=1 TO S:Q$(22+L,22+L)=Q$(22-S+L,22-S
+L):POSITION L+3,11:PRINT Q$(22+L,22+L);
3350 Q$(22-S+L,22-S+L)=" ":POSITION 35-S+L,9:P
RINT " ";:NEXT L
```

Line 3190 turns on the cursor and places it at the first location on the question line. Line 3195 checks to see if the new number of questions (N) is greater than or equal to 100. If so, the program transfers to lines 3600-3610 (as above). If N is less than 100, line 3200 resets the number of characters in the question (T) and the number of spaces at the end of the first line of the question (S) to zero.

Line 3210 sets R (the line number to which the program will return) to 3220. (This means that when a GOTO R is reached, the program will return to line 3220.) Then, line 3210 transfers control to a routine at lines 3700-3730. This routine first calls the control key routine at line 6610. After the control key routine is executed, control is transferred back to the calling routine at line 3700, which is a control key handling routine for question and answer entry.

```
3700 GOSUB 6610:IF FL=1 THEN 3730
3710 IF K=168 THEN Q$=B$:A$=B$(1,25):GOSUB 651
0:GOTO 3190
3720 IF K=170 THEN CLOSE #5:OPEN #5,12,0,DIR$:
```

```
N=N-1:GOSUB 6910:PRINT #5;N$:CLOSE #5:GOTO 401
0
3730 GOTO R
```

The code for entering data is also used by the edit section of the program. Therefore, FL is a flag that indicates if the entry routine is currently being used by the edit routine. A flag is simply a two-position switch, which can help us keep track of the status of program execution.

If FL (flag) = 1 (indicating the entry routine has been called by the editing routine), then the next two program lines (lines 3710 and 3720) are skipped because control keys are not allowed when we are editing a question.

If K = 168 (which indicates that ^R, restart, was pressed), line 3710 blanks the question and answer and clears a portion of the screen using the blanking routine at line 6510. Line 3710 then transfers control back to line 3190 to start entering the question again (restart).

If K = 170 (which indicates that ^E, exit, was pressed) line 3720 is executed. It closes the file and opens it again at the beginning of the file. Line 3720 then calls the number length routine at 6910. Upon return from this routine, line 3720 prints the number of questions to the file, closes the file, and returns control to the Edit Menu.

Line 3730 returns to location R (line number R as specified earlier, line 3220).

If <RETURN> is pressed and FL = 1 (indicating this is a question edit), line 3220 transfers control back to the edit options at line 4610. If <RETURN> is pressed (FL <> 1, which indicates that this is a new entry), line 3230 transfers control to line 3450 to enter the answer.

If <BACKSPACE> is pressed and T (number of characters) > 0, line 3240 prints a backspace. Then the character that was backspaced over is blanked from the question, T is decremented (one is subtracted from T), and control is transferred to line 3210 to get a new keypress.

Line 3270 checks to make sure that when K2 (the ATASCII keypress) equals 32 (a space), K (the key's scan code) equals 33 (scan code for a space). This is done to ensure that if the returned ATASCII keypress is 32, the space bar was pressed and not a special key, such as <TAB> or <CLEAR>. (See the Initialization section in Chapter 5.) If K2 doesn't equal 32 and K

doesn't equal 33, control is transferred to line 3210 to get another keypress.

Line 3280 increments the number of characters (T), adds the keypress (CHR$(K2)) to the question string (Q$), and prints the character to the screen.

Line 3300 checks to see if you are at the last character on the first line of the question (T = 22). If not, control transfers to 3210 for another keypress.

If you are at the end of the first question line, lines 3310-3350 are executed. Line 3310 checks to see if the last character typed was a space. If it was a space, the cursor is moved to the second line of the question and control is transferred to 3360 for input of the second line of the question.

If the last character typed wasn't a space, the word you are typing won't fit on the first line, and must be moved to the second line. The following occurs. Lines 3320-3330 count back through the number of characters until a space is reached. Line 3335 adjusts S (number of spaces at the end of the first line) and T (number of characters in the question) to reflect the number of actual spaces that must be added to the end of the first line to force the partial word to the second line. It then turns off the cursor. Lines 3340-3350 contain a FOR...NEXT loop, which moves the partial word to the second line, inserts spaces into Q$ (the question), and displays the moved word on the second line.

To be sure, such attention to aesthetics is not mandatory, but I don't like wrapped-around words, and I'll bet you don't either!

Entering second line of the question

This section is almost identical to the code for entering characters on the first line of the question. The only differences are in how the program handles a <BACKSPACE> keypress and the last character on the line. Therefore, we will only describe the sections of code that perform these functions.

```
3360 POKE 752,0:POSITION T-18,11:PRINT " ←";
3370 R=3372:GOTO 3700
3372 IF K=12 AND FL=1 THEN 4610
3375 IF K=12 THEN 3450
3380 IF K<>52 THEN 3397
3385 IF T>S+22 THEN POSITION T-19,11:PRINT " ←
";:Q$(T,T)=" ":T=T-1:GOTO 3370
```

```
3390 POKE 752,1:FOR L=1 TO S:Q$(22-S+L,22-S+L)
=Q$(22+L,22+L):POSITION 35-S+L,9:PRINT Q$(22-S
+L,22-S+L);
3395 Q$(22+L,22+L)=" ":POSITION L+3,11:PRINT "
 ";:NEXT L:Q$(22,22)=" ":POKE 752,0:POSITION 3
5,9:PRINT " ←";
3396 S=0:T=21:GOTO 3210
3397 IF T=54 THEN 3370
3400 IF K2=32 AND K<>33 THEN 3370
3410 T=T+1:Q$(T,T)=CHR$(K2):POSITION T-19,11:P
RINT CHR$(K2);:GOTO 3370
```

Line 3360 turns on the cursor and positions it on the second line.

If <BACKSPACE> is pressed, the following occurs. If you were typing a word on the second line and pressed <BACKSPACE>, line 3385 checks to see if the word (or partial word) will now fit on the end of the first line. If the number of characters on the second line (T - 22) is greater than the number of spaces available on the end of the first line (S), line 3385 performs a "normal" backspace and control is transferred to 3370 to get the next keypress. If the number of characters on the second line is less than or equal to the number of spaces available on the end of the first line, the FOR...NEXT loop at lines 3390-3396 is executed. This loop moves the characters from the second line into the spaces at the end of the first line, updates Q$, and positions the cursor at the end of the first line. Line 3396 sets S (the number of spaces) = 0, sets T (the number of characters) = 21, and transfers control to line 3210 for first line input.

The second difference between first and second line question entry is that when you reach the end of the second line (T = 54), the only characters accepted are <RETURN> and <BACKSPACE>.

The preceding sections on wraparound control can have future applications in programs you might write. You may want to mark them for future detailed review.

Entering the answer

Coming up next is a description of the code for entering the answer into the data base. This section is similar to the section for entering the question, except the answer is entered on only one line (and not two). Again, only the differences are described.

When entering answers, <BACKSPACE> is always a "normal" backspace (and not handled as it is on the second line of question entry). Also, only 25 characters (and not 54 characters) can be entered.

```
3450 T=0:POSITION 10,17:PRINT " ";
3460 R=3470:GOTO 3700
3470 IF K=12 AND FL=1 THEN 4630
3480 IF K=12 THEN 3560
3490 IF K=52 AND T>0 THEN POSITION T+10,17:PRI
NT " ←";:A$(T,T)=" ":T=T-1:GOTO 3460
3500 IF T=25 THEN 3460
3510 IF K2=32 AND K<>33 THEN 3460
3520 T=T+1:A$(T,T)=CHR$(K2):POSITION T+10,17:P
RINT CHR$(K2);:GOTO 3460
3560 PRINT #5;Q$:PRINT #5;A$:N=N+1:GOSUB 6410:
Q$=B$:A$=B$(1,25):GOSUB 6510:GOTO 3190
```

One of the neat (or frustrating, depending on your point of view) things about computer programs is that they are never done. You can always think of more features to add. If you are so inclined after a first runthrough of this book, here are a couple of suggestions for improving the answer input routine. First, you could make sure that the answer is no more than two words long. Second, you could eliminate any occurences of articles such as "a," "an," or "the." In this way, the player would not have to type the article to get a correct answer. For example, suppose you enter "the Constitution" as the answer to a question in the data base program, and the program doesn't eliminate "the." A trivia game player would have to answer the question as "the Constitution," even though "Constitution" is also correct. Finally, you could add a section of code that would make sure that *some* answer (at least one character) is entered before saving the question and answer to disk.

That, then, comprises the code for the entering data section of the program. But, as noted earlier, nobody is perfect, and even after we have entered what we think is correct information, we sometimes will have to change it. Time to look at the editing process, and that is next.

Editing data

Pull out Flowchart 1-I for simultaneous reference, refer to Appendix A, and let's look at the logic for option #2 from the

Edit Menu, edit data. Subroutines are in Chapter 7. This option allows you to edit data that is already entered into the data base. You can edit the question and/or the answer.

The next section of code indexes the records, and prints the Question Entry screen and edit options. It then sets up the program to allow the use of the edit options. The code is:

```
4120 PRINT "{CLEAR}":IF N=0 THEN POSITION 11,9:PRINT
 "NO DATA IN FILE":POSITION 12,11:PRINT "PRESS
 ANY KEY"
4130 IF N=0 THEN GOSUB 6010:GOTO 4010
4140 POSITION 11,11:PRINT ".... PLEASE WAIT"
4150 OPEN #4,12,0,DIR$:INPUT #4,N$:N=VAL(N$):F
OR L=1 TO N:NOTE #4,X,Y:S(L)=X:B(L)=Y:INPUT #4
,Q$,A$:NEXT L
4170 M=1:POKE 752,1:PRINT "{CLEAR}":GOSUB 6210:POSIT
ION 13,3:PRINT "QUESTION EDIT"
4180 POSITION 2,21:PRINT " ^F:FWD ^R:REV ^J:JM
P ^C:CHG ^E:EXT "
4200 POINT #4,S(M),B(M):INPUT #4,Q$,A$:N$=STR$
(M):IF M<100 THEN N$(3,3)=" ":IF M<10 THEN N$(
2,2)=" "
4210 POSITION 13,5:PRINT N$;:GOSUB 6810:GOSUB
6820
```

Line 4120 clears the screen and checks to see if there are no questions in the file (N = 0). If N = 0, the rest of line 4120 and line 4130 print a "no data in file" message, prompt to continue, and return control to the Edit Menu section of the program.

If N > 0 (there are questions in the file), the following lines are executed. Line 4140 prints a "please wait" message because indexing the records can take up to 22 seconds. A quick and easy method of customizing the program is to alter the "pacifier" message in line 4140. Line 4150 opens the file and uses a FOR...NEXT loop to record the sector and byte location of each question in variables S(100) and B(100). These are linear array variables, and can be thought of as a single column look-up table. Line 4170 sets M (the current question number) = 1 and calls the screen printing routine at 6210, which prints the Question Edit screen (see Fig. 6-3). Line 4180 prints the edit options.

Line 4200 points to the location on the disk of the current record; it then inputs that record into Q$ (question) and A$ (answer), and converts the current question number (M) into a string of length 3 (N$). Line 4210 prints the record number, the question, and the answer to the screen.

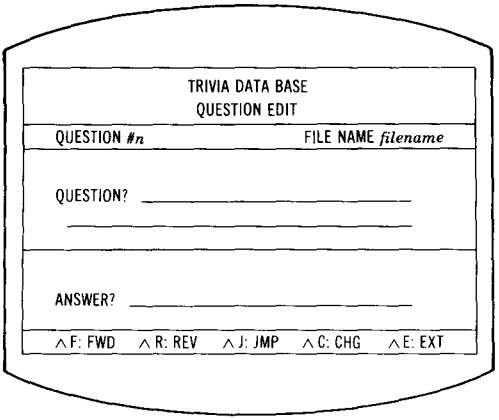


Fig 6-3. Question Edit screen.

Edit options

We now have several options: we can choose to search the file forward one record at at time, backwards one record at a time, jump to a record, change a record, or exit. See Fig. 6-3. But first we have to get your choice.

```
4220 GOSUB 6610:IF K=168 AND M>1 THEN M=M-1:GO
SUB 6510:GOTO 4200
4230 IF K=184 AND M<N THEN M=M+1:GOSUB 6510:GO
TO 4200
4240 IF K=129 THEN 4400
4250 IF K=146 THEN 4500
4260 IF K=170 THEN CLOSE #4:GOTO 4010
4270 GOTO 4220
```

Line 4220 calls the control key routine at line 6610, which gets your input.

The first option the program checks for is ^R:REV (reverse), which is the command to look at the previous record. After control is returned from the control key routine, line 4220 checks to see if K = 168 (^R) and if M (current record) > 1. If so, M is decremented by one, the screen is blanked by the blanking routine at 6510, and control is transferred to line 4200 to print the current information (question, answer, and question number) to the screen.

Likewise, we can look at the next record. This option is the ^F:FWD (forward) command. If K = 184 (^F) and M < N, line 4230 increments M by one, blanks the screen, and control is transferred to line 4200, which prints the new (current) information to the screen.

Another option is to "jump" to a specified record. This is the ^J:JMP command. With up to 100 records in a file, it would take forever to step through from the beginning to the end (i.e., if you were at record 2, and wanted to go to record 88); thus, the jump option. If K = 129 (^J), line 4240 transfers control to line 4400, which is a jump routine. The jump routine is described near the end of this chapter.

We can also change a record. The command for this is ^C:CHG. We can then redo the question (^Q:QUESTION) or the answer (^A:ANSWER). If K = 146 (^C), line 4250 transfers control to line 4500. This section of code will also be described shortly.

Finally, we can choose to exit this module and return to the Main Menu. The command for this function is ^E:EXT. If K = 170 (^E), line 4260 closes the file and transfers control back to the Edit Menu (line 4010).

If K was not equal to a valid edit option, then line 4270 transfers control back to line 4220 to get another keypress.

Now on to the two portions of code that are executed from the edit options section. They are the jump routine and the change option.

As described earlier, if you choose the ^J (jump) option, then K = 129. Line 4240 transfers control to the jump routine at line 4400.

```
4400 TRAP 4400:POSITION 3,21:PRINT B$(1,36);:P
OSITION 2,21:PRINT " JUMP TO WHICH RECORD ";:I
NPUT M
4410 IF M<1 OR M>N THEN 4400
4420 POSITION 38,21:PRINT CHR$(124);:TRAP 7010
:GOTO 4180
```

Line 4400 first sets the error trap to line 4400 in case there is an entry error. It blanks the bottom line on the screen and prompts the user to input the number of the record to jump to (M). Line 4410 makes sure that M is not less than 1 or greater than N. If M < 1 or M > N, control is transferred to 4400 to get another record number to jump to. Otherwise M is valid; line 4420 resets the

error trap to line 7010 and transfers control to 4180 to update the screen.

If you choose the ^C (change) option, then K = 146. Line 4250 transfers control to line 4500.

```
4500 POKE 752,1:POSITION 2,21:PRINT "  ^Q:QUES
TION  ^A:ANSWER  ^E:EXIT  "
4510 GOSUB 6610:IF K=175 THEN 4600
4520 IF K=191 THEN 4620
4530 IF K<>170 THEN 4510
4540 GOTO 4180
4600 POSITION 2,21:PRINT B$(1,35):Q$=B$:GOSUB
6810:FL=1:GOTO 3190
4610 FL=0:POINT #4,S(M),B(M):PRINT #4;Q$:PRINT
 #4;A$:GOTO 4500
4620 POSITION 2,21:PRINT B$(1,35):A$=B$(1,25):
GOSUB 6820:FL=1:POKE 752,0:GOTO 3450
4630 FL=0:POINT #4,S(M),B(M):PRINT #4;Q$:PRINT
 #4;A$:GOTO 4500
```

Line 4500 turns off the cursor and prints the change options on the screen. See Fig. 6-4. Line 4510 calls the control key routine at line 6610.

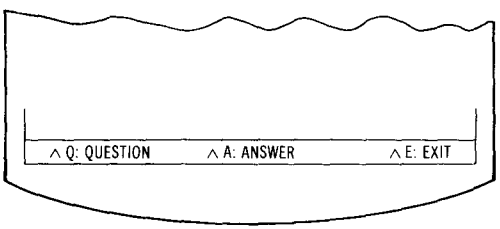


Fig. 6-4. Change options.

If K = 175 (^Q, question edit) upon returning from the control key routine, line 4510 transfers control to line 4600. Line 4600 blanks the question, sets FL = 1 (to signify an edit), and transfers control to line 3190 for question entry. Upon return from the question entry section, line 4610 updates the file with the new question and transfers control back to the change options (line 4500).

However, if K = 191 (^A, answer edit) upon returning from the control key routine, line 4520 transfers control to line 4620. Line 4620 blanks the answer, sets FL = 1 (to signify an edit), and

transfers control to line 3450 for answer entry. Upon return from the answer entry section, line 4630 updates the file with the new answer and transfers control back to the change options (line 4500).

If K <> 170 (not ^E, exit) upon returning from the control key routine, line 4530 transfers control back to line 4510 to get a different key because the keypress was not a valid choice.

Otherwise, K = 170 (^E, exit), and line 4540 transfers control back to the edit options (line 4180).

Summary

In Chapter 5 we learned about the Main Menu. We described that menu, and the initial program flow for options #1 (load a file) and #2 (create a new file). In addition, options #3 (delete a file) and #4 (exit the program) were described.

This chapter outlined the program flow if you choose option #1 or #2 from the Main Menu. If either of these options is chosen, the program proceeds to the Edit Menu. We then described, in detail, the three options in the Edit Menu. They are options #1 (enter data), #2 (edit data), and #3 (return to the Main Menu).

In the next chapter (Chapter 7) we'll learn about the subroutines in the data base program.

Chapter 7

Data Base Subroutines

Subroutines are of great value to the programmer because quite often a particular action is repeated throughout a program. Rather than copy the same code over and over, that action is identified as a subroutine. Then when the action is required, the subroutine is invoked, or called, eliminating many lines of code. The key input routine, used extensively up to this point, is a prime example.

Key input routine

The key input routine gets an input from the keyboard and converts it to an ATASCII character, which can be used by the data base program. The code that accomplishes this is:

```
6000 REM *** KEY INPUT ROUTINE
6010 K=PEEK(764):IF K>127 THEN 6010
6020 POKE 764,255:K2=PEEK(K+KS):RETURN
```

Line 6010 PEEKs to memory location 764 and sets K (the variable for the keypress scan code) equal to the scan code of the last key pressed. If no key was pressed, K = 255. If K > 127, a control key or no key (which are not valid keypresses in this instance) was pressed and the routine calls itself (is repeated).

After a valid key is pressed (thus, K has a value less than or equal to 127), line 6020 places a value of 255 into location 764, which signifies that we have already read the keypress. It then adds K to our KS (keystart) and we PEEK to find the ATASCII code (K2) corresponding to the keystroke. Finally, line 6020 transfers control to the line which called the key input routine.

The commands PEEK and POKE are not mysterious actions with cute names. If you visualize the memory of your Atari as little compartments in a desk, PEEKing allows you to look at the current contents, and POKEing allows you to put something into the compartment. Now you can amaze your friends by talking computerese when you work the terms PEEK and POKE into your cocktail party conversation!

File counting routine

The file counting routine counts the number of data files on the disk. This routine is necessary because the information on the number of data files is used by other parts of the program. This code is as follows:

```
6100 REM *** FILE COUNTING ROUTINE
6110 TRAP 6150:F=0
6120 OPEN #4,6,0,"D:*.*"
6130 INPUT #4,DIR$:IF DIR$(11,13)<>"TDB" THEN
6130
6140 F=F+1:GOTO 6130
6150 CLOSE #4:IF PEEK(195)<>136 THEN 7010
6160 TRAP 7010:RETURN
```

Line 6110 changes the location where the program will continue if an error occurs from its original location at 7010 to line 6150 (the reason for this will be explained in a moment). It then sets the file counter (F) to zero.

Line 6120 opens the directory so it can be read. Lines 6130-6140 read the directory entries sequentially until the end of the last file is reached. If the directory listing has the file name extension "TDB," one is added to the file counter (F). The program then goes back to read the next listing. When the program reaches the end of the directory, an error occurs. But we have changed the location of the error trapping routine to line 6150, so the program transfers to this line. This is a forced error (remember, errors can be our programming friends). We use a forced error because the directory length is not known, and because we want this type of error to be handled differently than the other errors.

Lines 6150-6160 close the directory, double check to make sure there was an "end of file" error, reset the error trapping routine to line 7010, and return control back to the line that called the file counting routine.

Screen printing routine

The screen printing routine prints a "generic" screen, which is used by the Question Entry and Question Edit screens. After the program executes the screen printing routine, the line which called it displays the additional information that corresponds to the particular screen. For example, suppose the Question Entry section of the program calls this routine. Upon returning from the screen printing routine, the program displays the words

"Question Entry" and the command line (^R:Restart and ^E:Exit), which are those parts of the screen different from the Question Edit screen.

Following is the code for the screen printing routine:

```
6200 REM *** SCREEN PRINTING ROUTINE
6210 PRINT "{CLEAR}":POKE 752,1:POSITION 1,1:PRINT "
{UPPER LEFT}";:POSITION 1,22:PRINT "{LOWER LEFT}";
6220 FOR C=2 TO 37:POSITION C,1:PRINT "{HORIZ}";:POS
ITION C,22:PRINT "{HORIZ}";:NEXT C
6230 POSITION 38,1:PRINT "{UPPER RIGHT}";:POSITION 38,22:P
RINT "{LOWER RIGHT}";
6240 FOR R=2 TO 21:POSITION 1,R:PRINT CHR$(124
);:POSITION 38,R:PRINT CHR$(124);:NEXT R
6250 POSITION 12,2:PRINT "TRIVIA DATA BASE"
6260 POSITION 1,4:PRINT "{LEFT T}";:FOR C=2 TO 37:PRI
NT "{HORIZ}";:NEXT C:PRINT "{RIGHT T}"
6270 POSITION 3,5:PRINT "QUESTION #":POSITION
19,5:PRINT "FILE NAME ";F$
6280 POSITION 1,6:PRINT "{LEFT T}";:FOR C=2 TO 37:PRI
NT "{HORIZ}";:NEXT C:PRINT "{RIGHT T}"
6290 POSITION 3,9:PRINT "QUESTION? "
6300 POSITION 14,10:FOR C=1 TO 22:PRINT "{HORIZ}";:N
EXT C
6310 POSITION 4,12:FOR C=1 TO 32:PRINT "{HORIZ}";:NE
XT C
6320 POSITION 1,14:PRINT "{LEFT T}";:FOR C=2 TO 37:PR
INT "{HORIZ}";:NEXT C:PRINT "{RIGHT T}"
6330 POSITION 3,17:PRINT "ANSWER? "
6340 POSITION 11,18:FOR C=1 TO 25:PRINT "{HORIZ}";:N
EXT C
6350 POSITION 1,20:PRINT "{LEFT T}";:FOR C=2 TO 37:PR
INT "{HORIZ}";:NEXT C:PRINT "{RIGHT T}"
6360 RETURN
```

Line 6210 clears the screen, turns off the cursor, and prints the upper left-hand corner and lower left-hand corner symbols. Line 6220 uses a FOR...NEXT loop to draw a line across the top and bottom of the screen. Line 6230 draws the upper right-hand corner and lower right-hand corner symbols. Line 6240 draws vertical lines down the sides of the screen.

Line 6250 prints "Trivia Data Base." Line 6260 uses a FOR...NEXT loop to draw a line across the middle of the screen. Line 6270 prints "Question #" and "Filename," and then prints the actual file name (F$). Line 6280 uses a FOR...NEXT loop to draw a line across the screen.

Line 6290 displays "Question?". Lines 6300-6310 contain a FOR...NEXT loop that draws the two lines for the question. Line 6320 uses a FOR...NEXT loop to draw a line across the screen.

Line 6330 displays "Answer?". Line 6340 prints the line for the answer. Line 6350 uses a FOR...NEXT loop to draw a line across the screen. Line 6360 returns control to the line that called the screen printing routine.

Question number routine

This routine prints the number of questions to the screen, using the following code:

```
6400 REM *** QUESTION NUMBER ROUTINE
6410 POKE 752,1:POSITION 13,5:PRINT N;
6420 RETURN
```

Line 6410 turns off the cursor. Then using variable N (the number of questions), it displays the number of questions in the appropriate position on the screen. Line 6420 returns control to the line that called this routine.

Blanking routine

The blanking routine blanks the portion of the screen that is directly above the question and answer line. The code is:

```
6500 REM *** BLANKING ROUTINE
6510 POKE 752,1:POSITION 14,9:PRINT B$(1,22);:
POSITION 4,11:PRINT B$(1,32);:POSITION 11,17:P
RINT B$(1,25);
6520 RETURN
```

Line 6510 turns off the cursor and blanks the screen above the question and answer lines by printing a blank string (B$). Control is then transferred back to the line that called this routine in line 6520.

Control key routine

The control key routine gets an input from the keyboard and converts it to an ATASCII character, which can be used by the data base program. This code is similar to the code for the key input routine, except this routine also allows the input of control keys. The code that accomplishes this is:

```
6600 REM *** CONTROL KEY ROUTINE
6610 K=PEEK(764):IF K=255 THEN 6610
```

```
6620 POKE 764,255:K2=PEEK(K+KS)
6630 RETURN
```

Line 6610 PEEKs to memory location 764 and sets K (the variable for the keypress scan code) equal to the scan code of the last key pressed. If no key was pressed, K is equal to 255 and the routine calls itself (is repeated).

After a key has been pressed (thus, K has a value not equal to 255), line 6620 puts a value of 255 into location 764 to signify that we have already read the keypress. Then line 6620 adds K to our KS (keystart) and PEEKs to find the ATASCII code (K2) corresponding to the keystroke. Line 6330 transfers control to the line that called the control key routine.

File name routine

The file name routine converts the file name you chose when you loaded an old file, created a new file, or deleted an old file to a file name that can be used by the data base program. The code is:

```
6700 REM *** FILE NAME ROUTINE
6710 DIR$(1,2)="D:":DIR$(3,LEN(F$)+3)=F$:DIR$(
LEN(F$)+3,LEN(F$)+7)=".TDB":DIR$(LEN(F$)+7,20)
="                    "
6720 RETURN
```

Line 6710 converts F$ into a valid name (DIR$) that can be used to open the data file. Line 6720 transfers control to the line that called the file name routine.

Question print routine

The question print routine is actually two short routines that print the question or the answer to the screen. These two routines are:

```
6800 REM *** QUESTION PRINT ROUTINE
6810 POKE 752,1:POSITION 14,9:PRINT Q$(1,22);:
POSITION 4,11:PRINT Q$(23,54);:RETURN
6820 POKE 752,1:POSITION 11,17:PRINT A$;:RETUR
N
```

Line 6810 turns off the cursor, prints both lines of the question (Q$) to the screen, and then transfers control to the line that called the question print routine.

Line 6820 turns off the cursor, prints the answer (A$) to the screen, and then transfers control to the line that called this routine.

Number length routine

The number length routine converts the number of questions in a file into a string variable that is three characters long. (A string variable can be identified by the fact that it ends with a "$" character. Simply put, a string variable can contain alphabetic or numeric characters that are not interpreted numerically. Another type of variable is numeric. This type of variable can't have letters or words assigned to it. Thus, a numeric string assigned a value of "1" can be used for calculations, whereas a string variable assigned a value of "1" regards the "1" not as a quantity but as a character. But, I digress). The string variable is three characters long because the first record of the file is always three spaces long; if we print a number out to the file that is less than three characters, the file structure will be destroyed. The code for the number length routine follows:

```
6900 REM *** NUMBER LENGTH ROUTINE
6910 N$=STR$(N):IF N<100 THEN N$(3,3)=" ":IF N
<10 THEN N$(2,2)=" "
6920 RETURN
```

Line 6910 converts N, the number of questions, to N$, the string representation of the number of questions. Line 6920 transfers control back to the line that called the number length routine.

Error trapping routine

This routine "traps" any execution errors, except for forced errors or input errors (where the error trap has been temporarily set to a different line number).

```
7000 REM *** DISK ERROR ROUTINE
7010 TRAP 7060:POKE 752,1:PRINT "T":E=PEEK(195
):POSITION 7,6:PRINT "ERROR ";E;" AT LINE ";PE
EK(186)+PEEK(187)*256
7020 IF E=162 THEN POSITION 14,8:PRINT "DISK F
ULL":GOTO 7040
7030 IF E>18 THEN POSITION 13,8:PRINT "DISK ER
ROR"
7040 POSITION 12,11:PRINT "PRESS ANY KEY":POKE
 764,255:GOSUB 6010:CLOSE #4:CLOSE #5
```

```
7050 IF E=144 THEN 8010
7060 RUN
```

Line 7010 first sets an error trap that will re-run the data base program if there is an error while this routine is running. It then prints the error number (E) and the line where the error occurred. If E = 162, then it's a "disk full" error and line 7020 prints "disk full." If E > 18, then it's a "disk error" and line 7030 prints "disk error." Line 7040 then prompts to continue the program.

If E = 144, it's a "disk not present" or "disk write-protected" error and line 7050 causes the program to end. If it's not either error, line 7060 re-runs the program. Consult your Atari manual for actual error codes to determine the problem.

This completes the code explanation for the data base program. Next, we will look at the random inquiry and scorekeeping program (the trivia game) using the same techniques employed in the preceding chapters. We're having some fun now!

Chapter 8

Game Analysis

Now we'll examine the trivia game program in detail. Refer to Apppendix A and Flowchart 2-A. Subroutines are described in Chapter 9.

Initialization

The beginning of the trivia game program is the initialization. This section of code sets up the key conversion routine, and initializes the variables and error trapping.

```
100 DIM KEY1$(64),KEY2$(64)
110 KEY1$="LJ;  K+*0 PU I-=V C  BXZ4 36 521, .
N M/ R EY TWQ9 07 8<>FHD  GSA"
120 KEY2$="LJ:  K\^0 PU I_[V C  BXZ$ #& % ![ ]
N M? R EY TWQ( )' @  FHD  GSA"
130 KS=PEEK(141)*256+PEEK(140)
140 POKE KS+94,34
150 DIM DIR$(20),Q$(54),A$(25),B$(54),F$(9),S(
100),B(100),G$(25),N$(3)
160 DIM NM1$(10),NM2$(10),NM3$(10),NM4$(10),NM
$(40),P(4)
170 FOR L=1 TO 54:B$(L,L)=" ":NEXT L:Q$=B$:A$=
B$(1,25):G$=A$:NM$=B$(1,40)
180 GRAPHICS 0:TRAP 7010
190 POKE 16,64:POKE 53774,64
```

Now, pull out Flowchart 2-B. We first set up the key conversion routine (lines 100-140). (See the Initialization section in Chapter 5 for a more complete description.) Line 100 dimensions 128 spaces in memory. ATASCII character are put into the space (lines 110-120); their position is determined by their Atari keyboard scan codes. Line 130 loads the starting location of these dimensioned characters in variable KS (the memory location where ATASCII characters are stored). This is done because the memory location of an ATASCII character can be derived by adding its scan code (K) to KS and PEEKing into memory. We have to do this because the scan codes do not follow in logical order. Line 140 places the ATASCII representation for a quote mark into the reserved area.

Lines 150-170 initialize the string and array variables that are used by the program. The string variables must be initialized to blanks because this is not done automatically by the Atari.

Line 180 sets the screen defaults to mode 0 and turns on the error trapping routine at line 7010. Line 190 disables the break key. This ensures that the program will not be stopped accidentally.

Choosing a file

When the program begins, we are presented with the File Choice Menu (see Fig. 8-1 and Flowchart 2-C).

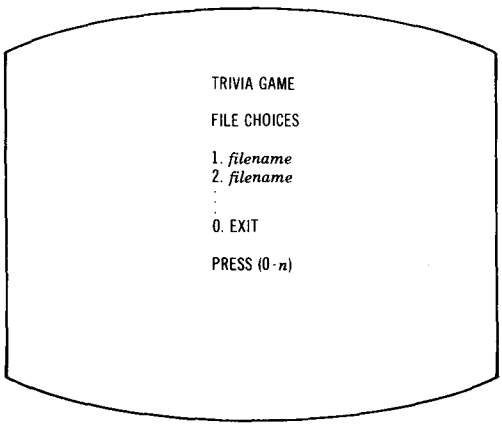


Fig. 8-1. File Choice Menu.

The code that displays the menu and allows the file choice is:

```
1000 REM *** FILE CHOICE MENU
1010 POKE 752,1:PRINT "{CLEAR}":POSITION 12,2:PRINT
"TRIVIA GAME"
1015 POSITION 12,4:PRINT "FILE CHOICES":GOSUB
6110:IF F>9 THEN F=9
1020 IF F=0 THEN 1200
1030 OPEN #4,6,0,"D:*.TDB":FOR L=1 TO F:INPUT
#4,DIR$:POSITION 12,L+5:PRINT L;". ";DIR$(3,10
):NEXT L:CLOSE #4
1035 POSITION 12,F+7:PRINT "0. EXIT"
1040 POSITION 12,F+9:PRINT "PRESS (0-";F;")"
1050 GOSUB 6010:IF K2<48 OR K2>F+48 THEN 1050
1055 IF K2=48 THEN 8010
1060 OPEN #5,6,0,"D:*.TDB"
```

```
1070 FOR L=1 TO K2-48:INPUT #5,DIR$:NEXT L:CLO
SE #5:F$=DIR$(3,11)
1080 IF F$(LEN(F$),LEN(F$))=" " THEN F$=F$(1,L
EN(F$)-1):GOTO 1080
1090 GOSUB 6710:OPEN #5,4,0,DIR$:INPUT #5;N$:C
LOSE #5:N=VAL(N$):IF N<20 THEN 1250
1100 GOTO 2010
1200 POSITION 8,6:PRINT "NO DATA FILES ON DISK
":POSITION 5,9:PRINT "YOU MUST SWITCH DATA DIS
KS"
1210 POSITION 8,10:PRINT "AND TYPE 'RUN' AGAIN
":POSITION 17,12:PRINT "OR"
1220 POSITION 12,14:PRINT "LOAD AND RUN":POSIT
ION 8,15:PRINT "THE DATA BASE PROGRAM"
1230 POSITION 12,19:PRINT "PRESS ANY KEY":GOSU
B 6010:GOTO 8010
1250 PRINT "¶":POSITION 5,8:PRINT "TOO FEW QUE
STIONS IN THE FILE":POSITION 12,9:PRINT "TO PL
AY A GAME"
1260 POSITION 12,11:PRINT "PRESS ANY KEY":GOSU
B 6010:GOTO 1010
```

First, line 1010 clears the screen and displays "Trivia Game." Line 1015 displays "File Choices," calls the routine at line 6110 to count the number of files on the disk, and sets the maximum number of files to nine.

If there are no files on disk (F = 0), line 1020 transfers control to lines 1200-1230. This section displays a "no files" message, gives you the options at this point, waits for a keypress to continue, and transfers control to line 8010 to exit the program. If there are one or more files on disk, line 1030 opens the directory for file names with the extension "TDB" and uses a FOR...NEXT loop to display these file names. It then closes the directory.

Lines 1035-1040 print the exit option on the screen and the prompt message to press a key.

Line 1050 calls the key input routine at line 6010 to get your choice. It then checks the validity of the keypress. If your choice is invalid (i.e., not one of the choices displayed) the key input routine is again called. If you pressed option 0, line 1055 transfers control to line 8010 to exit the program.

If your choice is valid and your input is not zero, the following lines are executed. Line 1060 re-opens the directory. Line 1070 reads in the file name you chose (F$), and line 1080 removes the

blanks in the file name. Line 1090 calls the file name routine at line 6710.

Upon return from the routine, line 1090 opens the data file, inputs the number of questions, and checks to see if N (the number of questions) is greater than 19. If it's not, control is transferred to lines 1250-1260, which print a "too few questions to play game" message, prompt the user to press a key to continue, and then return to the File Choice Menu. If N is greater than 19, line 1100 transfers control to line 2010 for name entry.

Entering names

After you choose a file, the trivia game program prompts you to enter the players' names.

The code for entering names is:

```
2000 REM *** NAME ENTRY
2010 POKE 752,1:PRINT "{CLEAR}"
2020 POSITION 13,2:PRINT "TRIVIA GAME"
2030 POSITION 4,5:PRINT "HOW MANY PLAYERS?  PR
ESS 1 TO 4"
2040 GOSUB 6010:P=K2-48:IF P<1 OR P>4 THEN 204
0
2050 POSITION 2,20:PRINT "NAMES MAY ONLY BE 10
 CHARACTERS LONG":POKE 752,0
2060 TRAP 2060:POSITION 3,8:PRINT "ENTER PLAYE
R 1'S NAME ";:INPUT NM1$:IF NM1$="" THEN 2060
2070 IF P=1 THEN 2130
2080 TRAP 2080:POSITION 3,10:PRINT "ENTER PLAY
ER 2'S NAME ";:INPUT NM2$:IF NM2$="" THEN 2080
2090 IF P=2 THEN 2130
2100 TRAP 2100:POSITION 3,12:PRINT "ENTER PLAY
ER 3'S NAME ";:INPUT NM3$:IF NM3$="" THEN 2100
2110 IF P=3 THEN 2130
2120 TRAP 2120:POSITION 3,14:PRINT "ENTER PLAY
ER 4'S NAME ";:INPUT NM4$:IF NM4$="" THEN 2120
2130 TRAP 7010:POKE 752,1:NM$(1,10)=NM1$:NM$(1
1,20)=NM2$:NM$(21,30)=NM3$:NM$(31,40)=NM4$
```

See Flowchart 2-D. Line 2010 turns off the cursor and clears the screen. Line 2020 prints the introductory message "Trivia Game." Line 2030 asks "How many players?" and prompts you to press a key from 1 to 4. Line 2040 calls the key input routine at line 6010, which gets your input. Upon returning from this

routine, line 2040 then converts the ATASCII keypress (K2) into a number of players variable (P), and checks to make sure P is between 1 and 4. If P isn't between 1 and 4, line 2040 calls the key input routine again. If P is between 1 and 4, line 2050 prints a message at the bottom of the screen that reminds the users that the players' names cannot be longer than 10 characters. It then turns on the cursor.

Line 2060 inputs the first player's name into variable NM1$; if NM1$ is blank, the line is re-run. If there is only one player, line 2070 transfers control to line 2130. Line 2080 places the second player's name into variable NM2$; if NM2$ is blank, the line is re-run. If there are only two players, line 2090 transfers control to line 2130. Line 2100 inputs the third player's name into variable NM3$; if NM3$ is blank, the line is re-run. If there are only three players, line 2110 transfers control to line 2130. Line 2120 puts the fourth player's name into variable NM4$; if NM4$ is blank, the line is re-run.

Each of the above input lines (2060, 2080, 2100, 2120) is trapped to itself in case of an input error. That is, if there is an input error, control will not transfer to the error trapping routine, instead the question will be asked again. Line 2130 resets the error trap to line 7010. It then turns off the cursor and stores all the names into one string called NM$.

Playing the game

Now on to the code for the actual play of the trivia game:

```
3000 REM *** GAME
3010 POKE 752,1:PRINT "{CLEAR}"
3020 POSITION 11,8:PRINT ".... PLEASE WAIT":PO
SITION 12,11:PRINT "INDEXING DATA"
3030 OPEN #4,12,0,DIR$:INPUT #4,N$:N=VAL(N$):F
OR L=1 TO N:NOTE #4,X,Y:S(L)=X:B(L)=Y:INPUT #4
,Q$,A$:NEXT L
3040 POSITION 10,14:PRINT "ORDERING QUESTIONS"
3050 FOR L=1 TO N:S(0)=S(L):B(0)=B(L):R=INT(RN
D(1)*N)+1:S(L)=S(R):B(L)=B(R):S(R)=S(0):B(R)=B
(0):NEXT L
3060 PRINT "{CLEAR}":GOSUB 6210:M=0:P(1)=0:P(2)=0:P(
3)=0:P(4)=0
3070 FOR L=1 TO P*5:GOSUB 6410
3075 FL=1:IF RND(1)<0.1 THEN FL=2:POSITION 10,
15:PRINT "BONUS POINT QUESTION"
```

cont. on next page

```
3080 M=M+1:IF M=P+1 THEN M=1
3090 POSITION 3,4:PRINT "GET READY ";NM$(M*10-
9,M*10):FOR D=1 TO 200:NEXT D
3100 POSITION 3,4:PRINT "GET SET ";NM$(M*10-9,
M*10);"  ":FOR D=1 TO 200:NEXT D
3110 POSITION 3,4:PRINT "GO ";NM$(M*10-9,M*10)
;"       "
3120 POINT #4,S(L),B(L):INPUT #4,Q$,A$:GOSUB 6
810
3130 T=0:Z=500:POKE 752,0:POSITION 10,12:PRINT
 " ";
3140 GOSUB 6610:IF Z=0 THEN 3200
3150 IF K=12 THEN 3200
3160 IF K=52 AND T>0 THEN POSITION T+10,12:PRI
NT " ←";:G$(T,T)=" ":T=T-1:GOTO 3140
3170 IF T=25 THEN 3140
3180 IF K2=32 AND K<>33 THEN 3140
3190 T=T+1:G$(T,T)=CHR$(K2):POSITION T+10,12:P
RINT CHR$(K2);:GOTO 3140
3200 POKE 752,1:IF G$<>A$ THEN 3220
3210 POSITION 7,15:PRINT "CORRECT - ";INT(Z*FL
/20);" POINTS SCORED":P(M)=P(M)+INT(Z*FL/20):G
OSUB 6410:GOTO 3300
3220 POSITION 10,15:PRINT "  INCORRECT ANSWER
 "
3300 FOR D=1 TO 200:NEXT D:POSITION 3,15:PRINT
 B$(1,34):G$=B$(1,25):GOSUB 6510:POSITION 33,4
:PRINT 25
3310 NEXT L
3320 POSITION 6,15:PRINT "GAME OVER --- PRESS
ANY KEY":POKE 764,255:GOSUB 6010:CLOSE #4
3330 PRINT "┌":POSITION 14,5:PRINT "RANKING"
3340 FOR L=1 TO P:POSITION 10,L+6:M=1:FOR L2=1
 TO P:IF P(L2)>P(M) THEN M=L2
3350 NEXT L2:PRINT NM$(M*10-9,M*10);"    ";P(M)
:P(M)=-1:NEXT L
3360 POSITION 12,P+8:PRINT "PRESS ANY KEY":POK
E 764,255:GOSUB 6010
3370 PRINT "┌":POSITION 5,5:PRINT "PLAY AGAIN?
  PRESS Y OR N"
3380 GOSUB 6010:IF K2=78 THEN 8010
3390 IF K2<>89 THEN 3380
3400 POSITION 2,8:PRINT "SAME PLAYERS AND FILE
?  PRESS Y OR N"
3410 GOSUB 6010:IF K2=78 THEN RUN
3420 IF K2<>89 THEN 3410
3430 OPEN #4,12,0,DIR$:GOTO 3040
```

Refer to Flowchart 2-E. First, line 3010 turns off the cursor and clears the screen. Lines 3020-3050 set up the data. Line 3020 prints a "please wait" message and an "indexing data" message.

Line 3030 opens the file and uses a FOR...NEXT loop to index the data by recording the sector and byte location of each question in variables S(100) and B(100). Line 3040 prints an "ordering questions" message. Here is another "pacifier" message, which you can easily customize with virtual impunity. Line 3050 uses a FOR...NEXT loop and the random number R to randomly order the questions. Then line 3060 clears the screen and prints the trivia Game screen (see Fig. 8-2). It then sets the current player number (M) to zero and resets the players' scores P(1), P(2), P(3), and P(4).

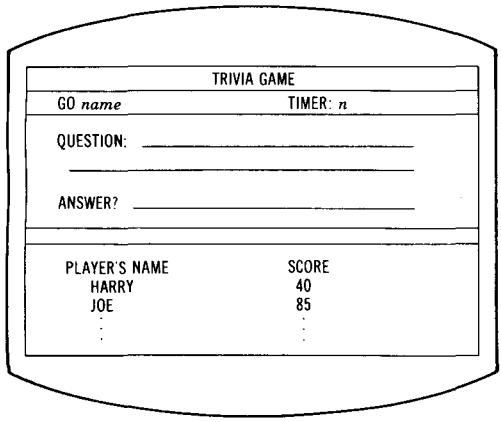


Fig. 8-2. Game screen.

Line 3070 starts the FOR...NEXT loop for 5 questions per player per game. It then calls the routine at line 6410, which is a score printing routine.

Line 3075 sets the flag (FL) = 1 for keeping score. Then, if the random number (RND(1)) is less than .1, FL is set to 2 (for double score), and a "bonus point" message is printed. Briefly stated, the function RND(1) will generate a random number between 0 and 1. If you want more frequent bonus questions, change the test on RND(1). As listed, 10% of the questions will be double score value. Line 3080 increments the current player number (M) and, if M is greater than P (number of players), then M is set to one.

Line 3090 prints a "get ready" message and pauses with a FOR...NEXT loop. Line 3100 prints a "get set" message and pauses with a FOR...NEXT loop. Line 3110 prints a "go" message. Line 3120 inputs the current (L) question (Q$) and answer (A$), and calls the routine at 6810 to print the question to the screen.

Line 3130 sets the number of characters in the guess (T) to zero, sets the timer variable (Z) to 500, turns the cursor on, and locates the cursor on the guess (answer) line. Line 3140 calls the control key timer routine at line 6610, which gets the keypress and keeps track of the time remaining. If Z = 0 (time has expired) upon return from the routine, line 3140 transfers control to line 3200.

If <RETURN> is pressed (K = 12), line 3150 transfers control to line 3200.

If <BACKSPACE> is pressed and T > 0 (number of characters in guess greater than zero), line 3160 prints a backspace. Then the character that was backspaced over is blanked from the guess, T is decremented, and control is transferred to line 3140 to get a new keypress.

Line 3170 only allows a <BACKSPACE> or <RETURN> if T = 25 (the guess line is full). Line 3180 checks to make sure that, when K2 (the ATASCII keypress) equals 32 (a space), K (the key's scan code) equals 33 (scan code for a space). This ensures that when the returned ATASCII keypress is 32, the space bar was pressed and not a special key, such as <TAB> (see the Initialization section in Chapter 5). If K2 doesn't equal 32 and K doesn't equal 33, control is transferred to line 3140 to get another keypress.

Line 3190 increments the number of characters (T), adds the keypress (CHR$(K2)) to the guess string (G$), and prints the character to the screen. It then transfers control to 3140 to get another keypress.

Line 3200 turns off the cursor and compares G$ and A$. If they are not equal, control is transferred to line 3220. Once you feel really comfortable with the existing code and get an ambitious urge, try modifying the code to allow for a beginner's and expert's game as follows. Parse (dissect) the input answer, and compare it with word one and word two of the correct answer. In a beginner's game, only one word has to match. For experts, it must be exactly correct. For example, let's say the correct answer is "John Brown." The beginner's version would give credit for

either word; the expert would have to have both answers correct. But back to the explanation of the code as written.

Line 3210 prints a "correct" and "points scored" message, and increases the player's score. It then updates the screen with the score printing routine (see Chapter 9), and transfers control to line 3300.

Line 3220 prints an "incorrect answer" message. Here is another candidate for code modification. You may wish to show the correct answer at this point. Line 3300 delays, blanks the message that was printed and the guess, and calls the blanking routine at line 6510. Line 3300 then prints the timer back to 25. Line 3310 is the point where the decision is made as to whether the FOR...NEXT loop is completed. It goes back to line 3070 if the loop has not been completed; line 3320 is executed if it has.

Line 3320 prints the "game over" message, prompts you to press a key to continue, and closes the file.

Line 3330 clears the screen and prints a "ranking" message. Lines 3340-3350 use a double FOR...NEXT loop to print the players' names and scores in the proper ranking (descending order).

Line 3360 prompts you to press a key to continue. Line 3370 clears the screen and prompts you to play again by pressing Y or N. Line 3380 calls the input routine and, if the choice is "N", exits the program. If the choice was not "Y", line 3390 calls the input routine again (because it was an invalid input).

Line 3400 ("Y" was pressed) prompts the user to see if the same names and file will be used. Line 3410 calls the input routine and, if the choice is "N", re-runs the program. If the choice was not "Y", line 3420 calls the input routine again (because it was an invalid input). Line 3430 re-opens the file and transfers control to line 3040 to re-order the questions and play again.

Exiting the program

And, at last, here's the code for ending the program:

```
8000 REM *** EXIT
8010 PRINT "{CLR}":POKE 16,192:POKE 53774,247:POKE
 752,0:END
```

Line 8010 clears the screen, turns on the cursor, enables the <BREAK> key, and ends the program.

This completes the explanation of the trivia game program. In the next chapter, we'll describe the subroutines used by the trivia game.

Chapter 9

Game Subroutines

By writing our subroutines as globally as possible, we are able to reuse many of them from the data base program in the game program. Any similarities are intentional.

Key input routine

The key input routine gets an input from the keyboard and converts it to an ATASCII character, which can be used by the trivia game program. The code that accomplishes this is:

```
6000 REM *** KEY INPUT ROUTINE
6010 K=PEEK(764):IF K>127 THEN 6010
6020 POKE 764,255:K2=PEEK(K+KS):RETURN
```

Line 6010 PEEKs to memory location 764 and sets K (the variable for the keypress scan code) equal to the scan code of the last key pressed. If no key was pressed, K = 255. If K > 127, a control key or no key (which are not valid keypresses in this instance) was pressed and the routine calls itself (is repeated).

After a valid key is pressed (thus, K has a value less than or equal to 127), line 6020 places a value of 255 into location 764, which signifies that we have already read the keypress. Then line 6020 adds K to our KS (keystart) and we PEEK to find the ATASCII code (K2) corresponding to the keystroke. Line 6020 then returns to the line that called the key input routine.

File counting routine

The file counting routine counts the number of data files on the disk. This routine is necessary because if there are no files on disk, you can't play the game. The code is:

```
6100 REM *** FILE COUNTING ROUTINE
6110 TRAP 6150:F=0
6120 OPEN #4,6,0,"D:*.*"
6130 INPUT #4,DIR$:IF DIR$(11,13)<>"TDB" THEN
6130
6140 F=F+1:GOTO 6130
6150 CLOSE #4:IF PEEK(195)<>136 THEN 7010
6160 TRAP 7010:RETURN
```

Line 6110 changes the location where the program will continue if an error occurs from its original location at 7010 to line 6150 (the reason for this will be explained in a moment). It then sets the file counter (F) to zero.

Line 6120 opens the directory so it can be read. Lines 6130-6140 read the directory entries sequentially until the end of the last file is read. If the directory listing has the file name extension "TDB", one is added to the file counter (F). The program then goes back to read the next listing. When the program reads the end of the last file, an error occurs. This causes the error trap at line 6150 to get program control. This is a forced error, which must be used because we don't know the length of the directory.

Lines 6150-6160 close the directory, double check to make sure there was an "end of file" error, reset the error trapping routine to line 7010, and return control back to the line that called the file counting routine.

Screen printing routine

This routine prints the Game screen using the following code:

```
6200 REM *** SCREEN PRINTING ROUTINE
6210 PRINT "▯":POKE 752,1:POSITION 1,1:PRINT "
▯";:POSITION 1,22:PRINT "▯";
6220 FOR C=2 TO 37:POSITION C,1:PRINT "▯";:POS
ITION C,22:PRINT "▯";:NEXT C
6230 POSITION 38,1:PRINT "▯";:POSITION 38,22:P
RINT "▯";
6240 FOR R=2 TO 21:POSITION 1,R:PRINT CHR$(124
);:POSITION 38,R:PRINT CHR$(124);:NEXT R
6250 POSITION 15,2:PRINT "TRIVIA GAME"
6260 POSITION 1,3:PRINT "▯";:FOR C=2 TO 37:PRI
NT "▯";:NEXT C:PRINT "▯"
6270 POSITION 25,4:PRINT "TIMER:  25"
6280 POSITION 1,5:PRINT "▯";:FOR C=2 TO 37:PRI
NT "▯";:NEXT C:PRINT "▯"
6290 POSITION 3,7:PRINT "QUESTION:"
6300 POSITION 14,8:FOR C=1 TO 22:PRINT "▯";:NE
XT C
6310 POSITION 4,10:FOR C=1 TO 32:PRINT "▯";:NE
XT C
6320 POSITION 1,14:PRINT "▯";:FOR C=2 TO 37:PR
INT "▯";:NEXT C:PRINT "▯"
6330 POSITION 3,12:PRINT "ANSWER?":POSITION 11
,13:FOR C=1 TO 25:PRINT "▯";:NEXT C
```

```
6340 POSITION 1,16:PRINT "├";:FOR C=2 TO 37:PR
INT "─";:NEXT C:PRINT "┤"
6350 POSITION 5,17:PRINT "PLAYER'S NAME","SCOR
E"
6360 POSITION 8,18:PRINT NM1$:POSITION 8,19:PR
INT NM2$:POSITION 8,20:PRINT NM3$:POSITION 8,2
1:PRINT NM4$
6370 RETURN
```

Line 6210 clears the screen, turns off the cursor, and prints the upper left-hand corner and lower left-hand corner symbols. Line 6220 uses a FOR...NEXT loop to draw a line across the top and bottom of the screen. Line 6230 draws the upper right-hand and lower right-hand corner symbols. Line 6240 draws vertical lines down the sides of the screen.

Line 6250 prints "Trivia Game." Line 6260 uses FOR...NEXT loops to draw a line across the middle of the screen. Line 6270 prints "Timer: 25". Line 6280 uses a FOR...NEXT loop to draw a line across the screen. Line 6290 displays "Question:". Lines 6300-6310 use a FOR...NEXT loop to draw the two lines for the question. Line 6320 uses a FOR...NEXT loop to draw a line across the screen. Line 6330 displays "Answer?" and the line for the guess. Line 6340 uses a FOR...NEXT loop to draw a line across the screen. Line 6350 prints "Player's Name" and "Score." Line 6360 prints the players' names (NM1$, NM2$, NM3$, and NM4$, to the screen. Line 6370 returns control to the line that called the screen printing routine.

Score printing routine

This routine updates the current score to the screen. The code follows:

```
6400 REM *** SCORE PRINTING ROUTINE
6410 POKE 752,1:POSITION 26,18:PRINT P(1):IF P
>1 THEN POSITION 26,19:PRINT P(2)
6420 IF P>2 THEN POSITION 26,20:PRINT P(3):IF
P>3 THEN POSITION 26,21:PRINT P(4)
6430 RETURN
```

Line 6410 turns off the cursor, prints the first player's score (P(1)), and if there is a second player, it prints the second player's score (P(2)). If there is a third player, line 6420 prints the third player's score (P(3)); if there's a fourth player, it also prints the fourth player's score (P(4)). Line 6430 returns control to the line that called this routine.

Blanking routine

The blanking routine blanks the portion of the screen that is directly above the question and answer line. The code is:

```
6500 REM *** BLANKING ROUTINE
6510 POKE 752,1:POSITION 14,7:PRINT B$(1,22);:
POSITION 4,9:PRINT B$(1,32);:POSITION 11,12:PR
INT B$(1,25);
6520 RETURN
```

Line 6510 turns off the cursor and blanks the screen above the question and answer lines by printing a blank string (B$). Line 6520 transfers control back to the line that called this routine.

Timer key routine

The timer key routine gets an input from the keyboard and converts it to an ATASCII character, which can be used by the trivia game. This code is similar to the code for the key input routine, except this routine keeps track of the elapsed time between when the question was printed and <RETURN> was pressed. The code that accomplishes this is:

```
6600 REM *** TIMER-KEY ROUTINE
6610 K=PEEK(764):Z=Z-1:POKE 752,1:POSITION 33,
4:PRINT INT(Z/20);"  ";:POKE 752,0:POSITION T+
11,12:PRINT " ←";
6620 IF Z=0 THEN 6650
6630 IF K=255 THEN 6610
6640 K2=PEEK(K+KS)
6650 POKE 764,255:RETURN
```

Line 6610 PEEKs to memory location 764 and sets K (the variable for the keypress scan code) equal to the scan code of the last key pressed. It then decrements the time counter (Z), turns off the cursor, and prints the time remaining (INT(Z/20)). Line 6610 then turns on the cursor and places it on the guess line. Line 6620 checks to see if Z = 0. If so, control is transferred to 6650. If no key was pressed, K = 255 and line 6630 re-calls the routine. The technical term for this condition is "running out of time."

After a key has been pressed (thus, K has a value not equal to 255), line 6640 adds K to our KS (keystart) and PEEKs to find the ATASCII code (K2) corresponding to the keystroke. Line 6650 POKEs a 255 into location 764 to signify that we have already read

the keypress, and transfers control to the line that called the timer key routine.

File name routine

The file name you chose to play the trivia game program has to be converted to a file name that can be used by the game. The file name routine does this conversion using the following code:

```
6700 REM *** FILE NAME ROUTINE
6710 DIR$(1,2)="D:":DIR$(3,LEN(F$)+3)=F$:DIR$(
LEN(F$)+3,LEN(F$)+7)=".TDB":DIR$(LEN(F$)+7,20)
="                    "
6720 RETURN
```

Line 6710 converts F$ into a valid name (DIR$) that can be used to open the data file. Line 6720 transfers control to the line that called this routine.

Question print routine

The question print routine prints the question to the screen. The code follows:

```
6800 REM *** QUESTION PRINT ROUTINE
6810 POKE 752,1:POSITION 14,7:PRINT Q$(1,22);:
POSITION 4,9:PRINT Q$(23,54);:RETURN
```

Line 6810 turns off the cursor and prints both lines of the question to the screen (Q$). Line 6810 transfers control to the line that called this routine.

Error trapping routine

This routine "traps" any execution errors, except for forced errors or input errors (where the error trap has been temporarily set to a different line number).

```
7000 REM *** DISK ERROR ROUTINE
7010 TRAP 7050:POKE 752,1:PRINT "┌":E=PEEK(195
):POSITION 9,6:PRINT "ERROR ";E;" AT LINE ";PE
EK(186)+PEEK(187)*256
7020 IF E>18 THEN POSITION 13,8:PRINT "DISK ER
ROR"
7030 POSITION 12,11:PRINT "PRESS ANY KEY":POKE
 764,255:GOSUB 6010:CLOSE #4:CLOSE #5
```

cont. on next page

```
7040 IF E=144 THEN 8010
7050 RUN
```

Line 7010 first sets an error trap that will re-run the trivia game program if there is an error while this routine is running. It then prints the error number (E) and the line where the error occurred. If E > 18, it's a disk error and line 7020 prints "disk error." Line 7030 then prompts to continue the program.

If it's a "disk not present" error or a "disk write-protected" error, E = 144 and line 7040 causes the program to end. If it's not either error, line 7050 re-runs the program. Consult your Atari manual for actual error codes to determine the problem.

This completes the code explanation for the trivia game program. Because we now have the data base program and game program under our belt, it's time to use them! So, let's look at the instructions for using the data base program (Chapter 10) and playing the game (Chapter 11). We're *really* going to have some fun now, boy.

Chapter 10

Using the Data Base

As noted earlier, the *Trivia Data Base* is the program used to store your questions and answers. You can have up to 100 questions per file, and usually up to 8 files on a disk. This chapter demonstrates how easy the program is to use.

The data base is totally menu driven. This means that you are able to select the process that you want to perform on the data base. Type **RUN "D:DATABASE"** and press <RETURN> to run the database program. (See the *Loading Instructions* section at the front of the book for more complete instructions.) Once the program is loaded, you are presented with the Main Menu. See Fig. 10-1.

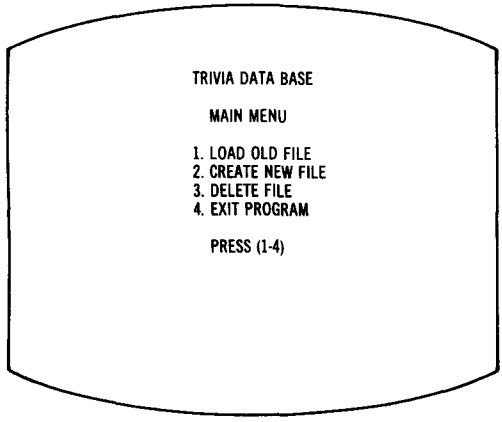


Fig. 10-1. Main Menu.

From this menu you select whether you want to load and work on a trivia file that has already been created with this program, create a new trivia file, delete a trivia file, or exit the data base program. Because this is your first time using the data base, you'll want to select option #2, Create New File, but before doing that, we'll explain each option.

#1. Load file

This option allows you to load a trivia file into memory so you are able to add new questions to the file or change questions and answers. Once this option is chosen, the File Choice Menu is displayed (see Fig. 10-2). After you choose a file, and the file is loaded, the Edit Menu is displayed. We will describe the Edit Menu options at the end of this chapter.

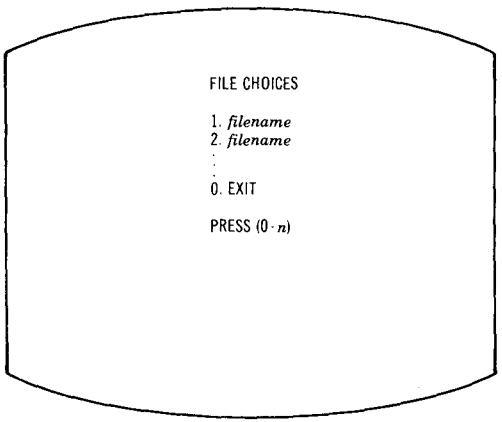


Fig. 10-2. File Choice Menu.

#2. Create new file

This option allows you to create a new file. A file can contain 100 questions and answers at most. Go ahead and select this option now. You will be prompted to enter the name of the file you want to create. *Important Note: file names can only contain capital letters!* After entering the file name, the Edit Menu is displayed.

We will look at each of the four selections given in Fig. 10-3 subsequently. For now, we will describe the two remaining options on the data base Main Menu.

#3. Delete file

To delete a file, you would choose option #3 from the Main Menu. After you have chosen this option, the File Choice Menu is displayed. Refer back to Fig. 10-2. Press the number corresponding to the file you want to delete. The screen will display a message for you to confirm that you indeed want to delete the

file (once it's gone, it's gone). If you press "N" the program returns to the Main Menu. If you press "Y" that file is deleted and the program returns to the Main Menu.

#4. Exit program

Selection #4 from the Main Menu ends the program. After you choose this option, the screen will display "Ready." This indicates that the computer is in Atari BASIC.

Enter and edit data

After you choose option #1 or #2 from the Main Menu, load a file or create a file, the Edit Menu is displayed. See Fig. 10-3. From the Edit Menu, you can enter new data into the data base, edit existing data, or return to the Main Menu.

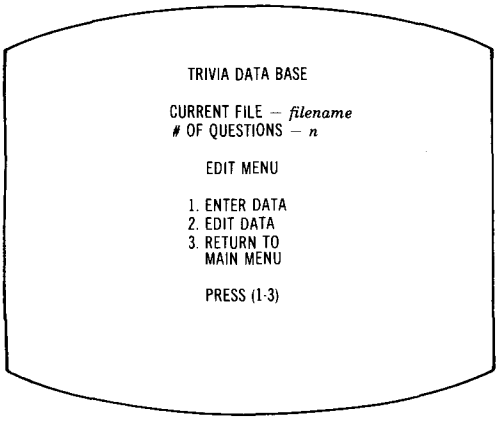


Fig. 10-3. Edit Menu.

#1. Enter data

This is the first thing you need to do when creating a new file. This selection allows you to enter new questions and answers to the trivia file. To let you know how many questions are in the file, the current question number is always displayed while you are adding new questions. The screen for adding the questions and answers is shown in Fig. 10-4.

First take a look at the bottom of the screen shown in Fig. 10-4. The bottom of this screen contains two keys. They are ^**R** (restart) and ^**E** (exit). The ^ stands for the <CONTROL> key.

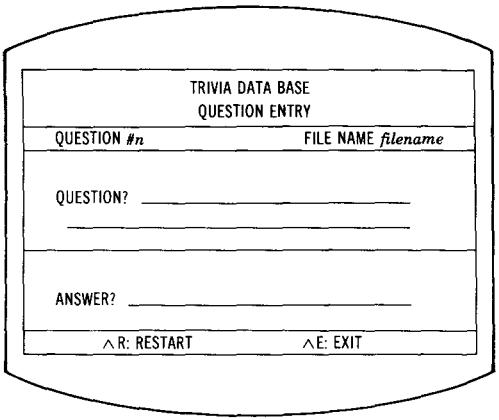


Fig. 10-4. Question Entry screen.

By pressing the <CONTROL> key and the <R> key simultaneously, you can restart the entry of the question and answer. In other words, if you are typing a question or an answer, and you want to begin over, **^R** erases what you have just typed and places the cursor at the beginning of the question line. The question or answer you were typing just before pressing **^R** is *not* saved to disk.

If you press **^E** (press the <CONTROL> key and the <E> key simultaneously), the program returns to the Edit Menu. If you were in the middle of typing a question or an answer, they are *not* saved to disk. You may press any one of the keys displayed at the bottom of the screen at any time.

Now you can enter your first question. Do not press <RETURN> to move the cursor from the first line of the question to the second. If you have a question that is more than one line long, just keep typing. The words will automatically "wrap around" to the second line, if necessary. When you are finished with your question, press the <RETURN> key to move the cursor to the answer line. The cursor then moves down to the answer field and waits for the answer.

When you have typed in the answer, press <Return> to add the question and answer to the data base (save them to disk). Once the question and answer have been saved, another blank screen is displayed and you can add more questions and answers.

When you have finished entering the questions and answers, press ^E as the first character on the question line. The program will return to the Edit Menu.

#2. Edit data

This routine allows you to display as well as edit (make changes) to any question or answer in the file. When you press option #2 from the Main Menu, a "please wait" message is displayed because the questions are being indexed, and this can take up to 22 seconds. Next, you'll see the Question Edit screen (Fig. 10-5).

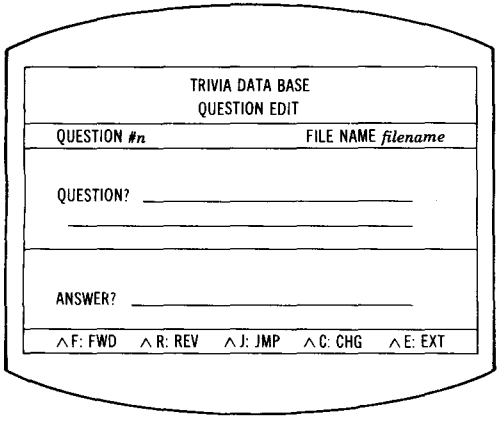


Fig. 10-5. Question Edit screen.

Again, as we noted in Fig. 10-4, there is a command line at the bottom of the screen. Here, there are five command keys to use. (Remember that the ^ stands for the <CONTROL> key, the letter just stands for the letter.) ^**F** (forward) displays the next question and answer. ^**R** (reverse) displays the previous question and answer. ^**J** (jump) jumps to a specified record.

^**C** (change) displays three additional commands, the change options (see Fig. 10-6). You can change the question (^**Q**), change the answer (^**A**), or exit (^**E**). When you are finished editing the question (or answer), press <RETURN> and the change options will again appear on the screen. Press ^**E** to exit and return to the Question Edit screen.

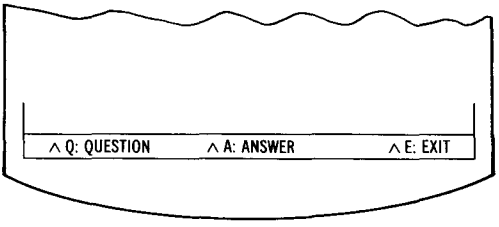


Fig. 10-6. Change options.

If you press ^E when you are in the Question Edit screen, the program returns to the Edit Menu.

#3. Return to Main Menu

Selection #3 allows you to quit working on the file currently in memory and return from the Edit Menu to the Main Menu. From the Main Menu, you can use an old file, create a new one, delete a file, or end the data base program.

Now that you understand how to enter and edit questions, try putting some questions and answers in a file so you can play the trivia game. If you bought the Combo Pack and really can't wait to play the game, go ahead and load the sample questions contained on the disk.

Chapter 11
Playing the Game

In Chapter 10 you learned how to enter questions and answers into your trivia data base file. This chapter uses the questions you entered into the file and allows you to play a trivia game. For those of you who did not store any questions and answers using the data base, you can use the sample file called QUIZ (if you bought the Combo Pack).

Overview

The game itself is much like any other trivia game. You are asked random questions and you have to answer them. The object of the game is to gain as many points as possible by correctly answering the questions. The faster you are able to answer a question, the more points you receive. The point value for each question starts at 25 points, but as time goes by the point value decreases. So it is to your benefit to answer the questions as fast as possible. There will be times when a question will be worth double points. These are the bonus questions. One important note: the answer you type must match the correct answer exactly; otherwise, it will not count as the correct answer.

The game can be played by one to four players. If a player misses a question, he does not receive any points and the next question is displayed.

The game keeps track of each player's name and current score. At the end of the game, the players' names and scores are shown in order, from highest to lowest.

Beginning the game

Type **RUN "D:GAME"** and press <ENTER> to run the trivia game. For more detailed information on loading the program, see the *Loading Instructions* section at the front of the book.

The first screen you see is the File Choice Menu (Fig. 11-1). From this menu, choose the number that corresponds to the file you want to use for the trivia game. You are asked how many

people plan to play. After entering the number of players, you enter their names. Then a "please wait" message is displayed while the program indexes the questions; and an "ordering questions" message is displayed while the program randomly orders the questions.

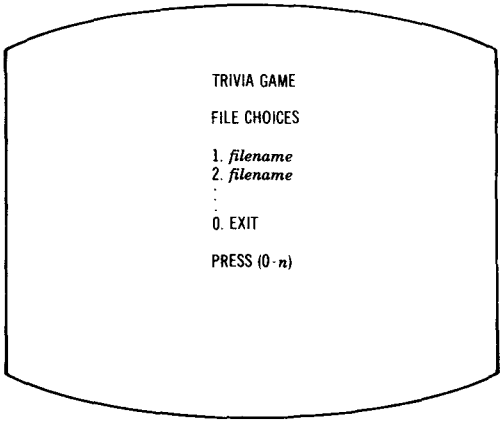


Fig. 11-1. File Choice Menu.

Playing the game

Once all the questions have been randomized, the game is ready to begin. The game screen is displayed (Fig. 11-2). Note that there is a timer in the upper right-hand section of the screen. This timer indicates how much time is remaining and also how much the question is worth. The status of all the players is at the bottom of the screen. It shows the player's name and current score.

When you press <RETURN> after entering your answer, the timer stops at the point value shown and the program checks your answer against the correct answer.

If you are right, the message "correct *n* points scored" is displayed (where *n* is the score), and you receive the number of points indicated by the timer. Some of the questions are randomly selected to be worth double the point value (the bonus questions). If it was a bonus question, you receive twice the points indicated by the timer. If you do not answer the question correctly, the next question is displayed.

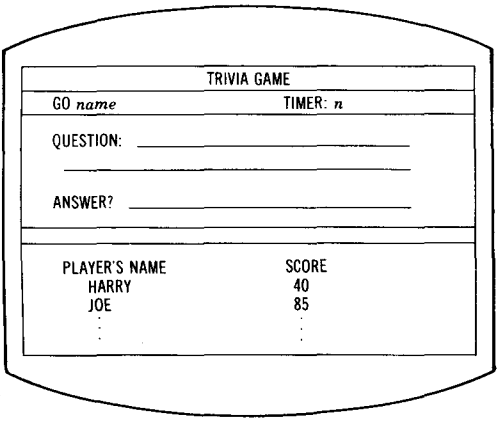


Fig. 11-2. Game screen.

Exiting the game

The game ends when five questions per player are asked. After the game ends, a screen is displayed that shows all the players' scores in descending (highest to lowest score) order. You are then asked if you wish to play another game. If you don't, the program exits. If you want to play again, you are asked if you want to use the same file and same players' names. If so, the questions are re-ordered and the game begins again. Otherwise, you start all over again by loading in another file.

The main point behind the trivia game is for you to enjoy the game while learning new and different trivia questions. Have fun!

Chapter 12
In Conclusion

Well, that's it. If you bought the Combo Pack, you have sample questions on the disk. We really strained our brains for those questions, but now it is up to you. If you have a copy of *Trivial Pursuit* (or some such game), you could enter some selected questions and play with the computer keeping score. If not, there are some other uses for the data base entry program and game program.

If you have students who need to drill on facts, they could have the questions entered (by you?), then practice with the computer's assistance. If you are studying for a professional exam, the same sort of assistance is available for you.

Whether you use these two programs for entertainment or education, it is our hope that you will gain from their purchase. Also, a thorough examination of the code structure and subroutines will be of great assistance in any other data base type programs that you might wish to write for your Atari computer.

Watch for other entertaining and instructional programs from us for your Atari.

Appendix A

Variable Descriptions

Data Base

String variables

KEY1$(64)	Reserves 64 spaces in memory to hold ASCII characters
KEY2$(64)	Reserves 64 spaces in memory to hold ASCII characters
DIR$(20)	Used to read in directory listings and to hold file names for opening data files
F$(8)	Current file name
Q$(54)	Current question
A$(25)	Current answer
B$(54)	Blank string that has 54 blanks, which are used to print blanks on the screen and blank the question and answer
N$(3)	Number of questions (string of length 3)

Array variables

S(100)	Sector locations of questions for indexing, S(1), S(2), S(3)...S(100)
B(100)	Byte locations of questions for indexing, B(1), B(2), B(3)...B(100)

Number variables

KS	Start of KEY1$ and KEY2$ in memory
K	Key scan code
L	Loop variable used in FOR...NEXT loops
F	Number of files on the disk
N	Number of questions in the file
T	Number of characters used in question and answer
S	Number of spaces at the end of the first line of the question

R	Line number to return to for question and answer entry
FL	Flag to signify whether an edit or new entry
M	Current question number in an edit
R	Loop variable to print horizontal lines on screen
C	Loop variable to print vertical lines on screen
E	Error code number

Game

String variables

KEY1$(64)	Reserves 64 spaces in memory to hold ASCII characters
KEY2$(64)	Reserves 64 spaces in memory to hold ASCII characters
DIR$(20)	Used to read in directory listings and to hold file names for opening data files
F$(8)	Current file name
Q$(54)	Current question
A$(25)	Current answer
B$(54)	Blank string that has 54 blanks, which are used to print blanks on the screen and blank the question and answer
N$(3)	Number of questions (string of length 3)
G$(25)	Player's guess
NM1$(10)	Player 1's name
NM2$(10)	Player 2's name
NM3$(10)	Player 3's name
NM4$(10)	Player 4's name
NM$(40)	A string that holds all the players' names in a single string

Array variables

S(100)	Sector locations of questions for indexing, S(1), S(2), S(3)...S(100)
B(100)	Byte locations of questions for indexing, B(1), B(2), B(3)...B(100)

P(4)	Player's scores, P(1), P(2), P(3), P(4)

Number variables

KS	Start of KEY1$ and KEY2$ in memory
K	Key scan code
L	Loop variable used in FOR...NEXT loops
F	Number of files on the disk
N	Number of questions in the file
P	Number of players
R	Random number used to order the questions
T	Number of characters used in question and answer
FL	Flag for bonus point questions
M	Number of the player for the current turn
Z	Number for timer
L2	Loop variable for nested FOR...NEXT loop
R	Loop variable to print horizontal lines on screen
C	Loop variable to print vertical lines on screen
D	Loop variable for delays
E	Error code number

Appendix B

Data Base Program Listing

This book has an accompanying disk (this book/disk combination is called a Combo Pack), which contains the program listings. If you did not buy the Combo Pack, you have to type the listings into the computer. But before doing so, there are a few things that are helpful to know.

First, the data base and game program listings contain certain symbols. These symbols are what is displayed when certain keys are pressed. Following is a list of all symbols that appear in the listings, and their corresponding keystrokes.

Symbol	Keys to press*
	<CONTROL> <A>
	<CONTROL> <C>
	<CONTROL> <D>
	<CONTROL> <E>
	<CONTROL> <M>
	<CONTROL> <Q>
	<CONTROL> <R>
	<CONTROL> <Z>
	<ESC> and <CONTROL> <+> Press and release the <ESC> key and then press <CONTROL> <+> simultaneously
	<ESC> and <CONTROL> <CLEAR/< >

*Hold down the <CONTROL> key while pressing the following key.

Also, lines 110 and 120 in the data base and game listings contain the key conversion routine. When you are typing these lines, be sure to type them *exactly* as shown.

```
100 DIM KEY1$(64),KEY2$(64)
110 KEY1$="LJ;  K+*O PU I-=V C  BXZ4 36 521, .
N M/ R EY TWQ9 07 8<>FHD  GSA"
120 KEY2$="LJ:  K\^O PU I_¦V C  BXZ$ #& % ![ ]
N M? R EY TWQ( )' @  FHD  GSA"
130 KS=PEEK(141)*256+PEEK(140)
140 POKE KS+94,34
150 DIM DIR$(20),Q$(54),A$(25),B$(54),F$(8),S(
100),B(100),N$(3)
160 FOR L=1 TO 54:B$(L,L)=" ":NEXT L:Q$=B$:A$=
B$(1,25)
170 GRAPHICS 0:TRAP 7010
180 GOSUB 6110
190 POKE 16,64:POKE 53774,64
200 REM *** MAIN MENU
210 POKE 752,1:PRINT "¶"
220 POSITION 11,2:PRINT "TRIVIA DATA BASE"
230 POSITION 14,5:PRINT "MAIN MENU"
240 POSITION 11,8:PRINT "1. LOAD OLD FILE"
250 POSITION 11,10:PRINT "2. CREATE NEW FILE"
260 POSITION 11,12:PRINT "3. DELETE FILE"
270 POSITION 11,14:PRINT "4. EXIT PROGRAM"
280 POSITION 14,17:PRINT "PRESS (1-4)"
290 GOSUB 6010:IF K2<49 OR K2>52 THEN 290
300 ON K2-48 GOTO 1010,2010,5010,8010
1000 REM *** FILE CHOICE MENU
1010 PRINT "¶":POSITION 12,3:PRINT "FILE CHOIC
ES"
1015 IF F>9 THEN F=9
1020 IF F=0 THEN 1100
1030 OPEN #4,6,0,"D:*.TDB":FOR L=1 TO F:INPUT
#4,DIR$:POSITION 12,L+5:PRINT L;". ";DIR$(3,10
):NEXT L:CLOSE #4
1035 POSITION 12,F+7:PRINT "0. EXIT"
1040 POSITION 12,F+9:PRINT "PRESS (0-";F;")"
1050 GOSUB 6010:IF K2<48 OR K2>F+48 THEN 1050
1055 IF K2=48 THEN 210
1060 OPEN #5,6,0,"D:*.TDB"
1070 FOR L=1 TO K2-48:INPUT #5,DIR$:NEXT L:CLO
SE #5:F$=DIR$(3,11)
1080 IF F$(LEN(F$),LEN(F$))=" " THEN F$=F$(1,L
EN(F$)-1):GOTO 1080
1090 GOSUB 6710:OPEN #5,4,0,DIR$:INPUT #5;N$:C
LOSE #5:N=VAL(N$):GOTO 4010
1100 POSITION 7,6:PRINT "NO OLD DATA FILES ON
DISK":POSITION 12,8:PRINT "PRESS ANY KEY"
1110 GOSUB 6010:GOTO 210
2000 REM *** NEW FILE CHOICE
2010 PRINT "¶":POKE 752,0:F$=""
```

```
2015 IF F>=9 THEN 2090
2020 POSITION 7,5:PRINT "NEW FILE NAME ";:INPU
T F$
2023 IF F$="" THEN 2070
2025 FOR L=1 TO LEN(F$):IF ASC(F$(L,L))<65 OR
ASC(F$(L,L))>90 THEN 2070
2027 NEXT L
2030 POKE 752,1:TRAP 2060:OPEN #5,6,0,"D:*.*"
2040 INPUT #5,DIR$:IF DIR$(3,LEN(F$)+2)=F$ THE
N CLOSE #5:GOTO 2070
2050 GOTO 2040
2060 CLOSE #5:TRAP 7010:GOSUB 6710:OPEN #5,8,0
,DIR$:N=0:N$="0   ":PRINT #5;N$:CLOSE #5:F=F+1:
GOTO 4010
2070 POKE 752,1:POSITION 3,8:PRINT "ILLEGAL OR
 DUPLICATE FILE NAME":POSITION 11,11:PRINT "PR
ESS ANY KEY"
2080 GOSUB 6010:GOTO 210
2090 POKE 752,1:POSITION 8,8:PRINT "TOO MANY F
ILES ON DISK":POSITION 12,11:PRINT "PRESS ANY
KEY"
2100 GOSUB 6010:GOTO 210
3000 REM *** QUESTION ENTRY
3010 IF N>=100 THEN 3600
3020 GOSUB 6210:POSITION 13,3:PRINT "QUESTION
ENTRY":N=N+1:GOSUB 6410
3030 POSITION 9,21:PRINT "^R:RESTART   ^E:EXIT
";
3180 OPEN #5,9,0,DIR$:Q$=B$:A$=B$(1,25)
3190 POKE 752,0:POSITION 13,9:PRINT " ";
3195 IF N>100 THEN N=100:GOTO 3600
3200 T=0:S=0
3210 R=3220:GOTO 3700
3220 IF K=12 AND FL=1 THEN 4610
3230 IF K=12 THEN 3450
3240 IF K=52 AND T>0 THEN POSITION T+13,9:PRIN
T " ←";:Q$(T,T)=" ":T=T-1:GOTO 3210
3270 IF K2=32 AND K<>33 THEN 3210
3280 T=T+1:Q$(T,T)=CHR$(K2):POSITION T+13,9:PR
INT CHR$(K2);
3300 IF T<22 THEN 3210
3310 IF K2=32 THEN POSITION 4,11:PRINT " ←";:G
OTO 3370
3320 FOR S=22 TO 1 STEP -1:IF Q$(S,S)=" " THEN
 3335
3330 NEXT S
3335 S=22-S:T=22+S:POKE 752,1
3340 FOR L=1 TO S:Q$(22+L,22+L)=Q$(22-S+L,22-S
+L):POSITION L+3,11:PRINT Q$(22+L,22+L);
```

```
3350 Q$(22-S+L,22-S+L)=" ":POSITION 35-S+L,9:P
RINT " ";:NEXT L
3360 POKE 752,0:POSITION T-18,11:PRINT " ←";
3370 R=3372:GOTO 3700
3372 IF K=12 AND FL=1 THEN 4610
3375 IF K=12 THEN 3450
3380 IF K<>52 THEN 3397
3385 IF T>S+22 THEN POSITION T-19,11:PRINT " ←
";:Q$(T,T)=" ":T=T-1:GOTO 3370
3390 POKE 752,1:FOR L=1 TO S:Q$(22-S+L,22-S+L)
=Q$(22+L,22+L):POSITION 35-S+L,9:PRINT Q$(22-S
+L,22-S+L);
3395 Q$(22+L,22+L)=" ":POSITION L+3,11:PRINT "
 ";:NEXT L:Q$(22,22)=" ":POKE 752,0:POSITION 3
5,9:PRINT " ←";
3396 S=0:T=21:GOTO 3210
3397 IF T=54 THEN 3370
3400 IF K2=32 AND K<>33 THEN 3370
3410 T=T+1:Q$(T,T)=CHR$(K2):POSITION T-19,11:P
RINT CHR$(K2);:GOTO 3370
3450 T=0:POSITION 10,17:PRINT " ";
3460 R=3470:GOTO 3700
3470 IF K=12 AND FL=1 THEN 4630
3480 IF K=12 THEN 3560
3490 IF K=52 AND T>0 THEN POSITION T+10,17:PRI
NT " ←";:A$(T,T)=" ":T=T-1:GOTO 3460
3500 IF T=25 THEN 3460
3510 IF K2=32 AND K<>33 THEN 3460
3520 T=T+1:A$(T,T)=CHR$(K2):POSITION T+10,17:P
RINT CHR$(K2);:GOTO 3460
3560 PRINT #5;Q$:PRINT #5;A$:N=N+1:GOSUB 6410:
Q$=B$:A$=B$(1,25):GOSUB 6510:GOTO 3190
3600 PRINT "▼":POKE 752,1:POSITION 14,8:PRINT
"FILE FULL":POSITION 12,11:PRINT "PRESS ANY KE
Y"
3610 GOSUB 6010:CLOSE #5:OPEN #5,12,0,DIR$:GOS
UB 6910:PRINT #5;N$:CLOSE #5:GOTO 4010
3700 GOSUB 6610:IF FL=1 THEN 3730
3710 IF K=168 THEN Q$=B$:A$=B$(1,25):GOSUB 651
0:GOTO 3190
3720 IF K=170 THEN CLOSE #5:OPEN #5,12,0,DIR$:
N=N-1:GOSUB 6910:PRINT #5;N$:CLOSE #5:GOTO 401
0
3730 GOTO R
4000 REM *** EDIT MENU
4010 POKE 752,1:PRINT "▼"
4020 POSITION 11,2:PRINT "TRIVIA DATA BASE"
4030 POSITION 9,4:PRINT "CURRENT FILE - ";F$:P
OSITION 9,5:PRINT "# OF QUESTIONS - ";N
```

```
4040 POSITION 14,8:PRINT "EDIT MENU"
4050 POSITION 11,11:PRINT "1. ENTER DATA"
4060 POSITION 11,13:PRINT "2. EDIT DATA"
4070 POSITION 11,15:PRINT "3. RETURN TO":POSIT
ION 14,16:PRINT "MAIN MENU"
4080 POSITION 13,19:PRINT "PRESS (1-3)"
4100 GOSUB 6010:IF K2<49 OR K2>51 THEN 4100
4110 ON K2-48 GOTO 3010,4120,210
4120 PRINT "{":IF N=0 THEN POSITION 11,9:PRINT
 "NO DATA IN FILE":POSITION 12,11:PRINT "PRESS
 ANY KEY"
4130 IF N=0 THEN GOSUB 6010:GOTO 4010
4140 POSITION 11,11:PRINT ".... PLEASE WAIT"
4150 OPEN #4,12,0,DIR$:INPUT #4,N$:N=VAL(N$):F
OR L=1 TO N:NOTE #4,X,Y:S(L)=X:B(L)=Y:INPUT #4
,Q$,A$:NEXT L
4170 M=1:POKE 752,1:PRINT "{":GOSUB 6210:POSIT
ION 13,3:PRINT "QUESTION EDIT"
4180 POSITION 2,21:PRINT " ^F:FWD ^R:REV ^J:JM
P ^C:CHG ^E:EXT "
4200 POINT #4,S(M),B(M):INPUT #4,Q$,A$:N$=STR$
(M):IF M<100 THEN N$(3,3)=" ":IF M<10 THEN N$(
2,2)=" "
4210 POSITION 13,5:PRINT N$;:GOSUB 6810:GOSUB
6820
4220 GOSUB 6610:IF K=168 AND M>1 THEN M=M-1:GO
SUB 6510:GOTO 4200
4230 IF K=184 AND M<N THEN M=M+1:GOSUB 6510:GO
TO 4200
4240 IF K=129 THEN 4400
4250 IF K=146 THEN 4500
4260 IF K=170 THEN CLOSE #4:GOTO 4010
4270 GOTO 4220
4400 TRAP 4400:POSITION 3,21:PRINT B$(1,36);:P
OSITION 2,21:PRINT " JUMP TO WHICH RECORD ";:I
NPUT M
4410 IF M<1 OR M>N THEN 4400
4420 POSITION 38,21:PRINT CHR$(124);:TRAP 7010
:GOTO 4180
4500 POKE 752,1:POSITION 2,21:PRINT "  ^Q:QUES
TION  ^A:ANSWER  ^E:EXIT  "
4510 GOSUB 6610:IF K=175 THEN 4600
4520 IF K=191 THEN 4620
4530 IF K<>170 THEN 4510
4540 GOTO 4180
4600 POSITION 2,21:PRINT B$(1,35):Q$=B$:GOSUB
6810:FL=1:GOTO 3190
4610 FL=0:POINT #4,S(M),B(M):PRINT #4;Q$:PRINT
 #4;A$:GOTO 4500
```

```
4620 POSITION 2,21:PRINT B$(1,35):A$=B$(1,25):
GOSUB 6820:FL=1:POKE 752,0:GOTO 3450
4630 FL=0:POINT #4,S(M),B(M):PRINT #4;Q$:PRINT
 #4;A$:GOTO 4500
5000 REM *** FILE DELETION
5010 PRINT "{CLEAR}":POSITION 12,3:PRINT "FILE DELET
ION"
5020 IF F=0 THEN 5120
5030 OPEN #4,6,0,"D:*.TDB":FOR L=1 TO F:INPUT
#4,DIR$:POSITION 12,L+5:PRINT L;". ";DIR$(3,10
):NEXT L:CLOSE #4
5035 POSITION 12,F+7:PRINT "0. EXIT"
5040 POSITION 9,F+9:PRINT "DELETE WHICH FILE?"
:POSITION 12,F+10:PRINT "PRESS (0-";F;")"
5050 GOSUB 6010:IF K2<48 OR K2>F+48 THEN 5050
5055 IF K2=48 THEN 210
5060 OPEN #4,6,0,"D:*.TDB":FOR L=1 TO K2-48:IN
PUT #4,DIR$:NEXT L:CLOSE #4:F$=DIR$(3,11)
5070 IF F$(LEN(F$),LEN(F$))=" " THEN F$=F$(1,L
EN(F$)-1):GOTO 5070
5080 PRINT "{CLEAR}":POSITION 12,10:PRINT "DELETE ";
F$;"?":POSITION 12,12:PRINT "PRESS Y OR N"
5090 GOSUB 6010:IF K2=78 THEN 210
5100 IF K2=89 THEN GOSUB 6710:XIO 33,#1,0,0,DI
R$:F=F-1:GOTO 210
5110 GOTO 5090
5120 POSITION 10,6:PRINT "NO FILES TO DELETE":
POSITION 12,8:PRINT "PRESS ANY KEY"
5130 GOSUB 6010:GOTO 210
6000 REM *** KEY INPUT ROUTINE
6010 K=PEEK(764):IF K>127 THEN 6010
6020 POKE 764,255:K2=PEEK(K+KS):RETURN
6100 REM *** FILE COUNTING ROUTINE
6110 TRAP 6150:F=0
6120 OPEN #4,6,0,"D:*.*"
6130 INPUT #4,DIR$:IF DIR$(11,13)<>"TDB" THEN
6130
6140 F=F+1:GOTO 6130
6150 CLOSE #4:IF PEEK(195)<>136 THEN 7010
6160 TRAP 7010:RETURN
6200 REM *** SCREEN PRINTING ROUTINE
6210 PRINT "{CLEAR}":POKE 752,1:POSITION 1,1:PRINT "
{UPPER LEFT}";:POSITION 1,22:PRINT "{LOWER LEFT}";
6220 FOR C=2 TO 37:POSITION C,1:PRINT "{HORIZ}";:POS
ITION C,22:PRINT "{HORIZ}";:NEXT C
6230 POSITION 38,1:PRINT "{UPPER RIGHT}";:POSITION 38,22:P
RINT "{LOWER RIGHT}";
6240 FOR R=2 TO 21:POSITION 1,R:PRINT CHR$(124
);:POSITION 38,R:PRINT CHR$(124);:NEXT R
```

```
6250 POSITION 12,2:PRINT "TRIVIA DATA BASE"
6260 POSITION 1,4:PRINT "╠";:FOR C=2 TO 37:PRI
NT "═";:NEXT C:PRINT "╣"
6270 POSITION 3,5:PRINT "QUESTION #":POSITION
19,5:PRINT "FILE NAME ";F$
6280 POSITION 1,6:PRINT "╠";:FOR C=2 TO 37:PRI
NT "═";:NEXT C:PRINT "╣"
6290 POSITION 3,9:PRINT "QUESTION? "
6300 POSITION 14,10:FOR C=1 TO 22:PRINT "═";:N
EXT C
6310 POSITION 4,12:FOR C=1 TO 32:PRINT "═";:NE
XT C
6320 POSITION 1,14:PRINT "╠";:FOR C=2 TO 37:PR
INT "═";:NEXT C:PRINT "╣"
6330 POSITION 3,17:PRINT "ANSWER? "
6340 POSITION 11,18:FOR C=1 TO 25:PRINT "═";:N
EXT C
6350 POSITION 1,20:PRINT "╠";:FOR C=2 TO 37:PR
INT "═";:NEXT C:PRINT "╣"
6360 RETURN
6400 REM *** QUESTION NUMBER ROUTINE
6410 POKE 752,1:POSITION 13,5:PRINT N;
6420 RETURN
6500 REM *** BLANKING ROUTINE
6510 POKE 752,1:POSITION 14,9:PRINT B$(1,22);:
POSITION 4,11:PRINT B$(1,32);:POSITION 11,17:P
RINT B$(1,25);
6520 RETURN
6600 REM *** CONTROL KEY ROUTINE
6610 K=PEEK(764):IF K=255 THEN 6610
6620 POKE 764,255:K2=PEEK(K+KS)
6630 RETURN
6700 REM *** FILE NAME ROUTINE
6710 DIR$(1,2)="D:":DIR$(3,LEN(F$)+3)=F$:DIR$(
LEN(F$)+3,LEN(F$)+7)=".TDB":DIR$(LEN(F$)+7,20)
="                    "
6720 RETURN
6800 REM *** QUESTION PRINT ROUTINE
6810 POKE 752,1:POSITION 14,9:PRINT Q$(1,22);:
POSITION 4,11:PRINT Q$(23,54);:RETURN
6820 POKE 752,1:POSITION 11,17:PRINT A$;:RETUR
N
6900 REM *** NUMBER LENGTH ROUTINE
6910 N$=STR$(N):IF N<100 THEN N$(3,3)=" ":IF N
<10 THEN N$(2,2)=" "
6920 RETURN
7000 REM *** DISK ERROR ROUTINE
```

```
7010 TRAP 7060:POKE 752,1:PRINT "{CLEAR}":E=PEEK(195
):POSITION 7,6:PRINT "ERROR ";E;" AT LINE ";PE
EK(186)+PEEK(187)*256
7020 IF E=162 THEN POSITION 14,8:PRINT "DISK F
ULL":GOTO 7040
7030 IF E>18 THEN POSITION 13,8:PRINT "DISK ER
ROR"
7040 POSITION 12,11:PRINT "PRESS ANY KEY":POKE
 764,255:GOSUB 6010:CLOSE #4:CLOSE #5
7050 IF E=144 THEN 8010
7060 RUN
8000 REM *** EXIT
8010 PRINT "{CLEAR}":POKE 16,192:POKE 53774,247:POKE
 752,0:END
```

Appendix C

Game Program Listing

If you are typing the listing for the game program into the computer, see the notes at the beginning of Appendix B.

```
100 DIM KEY1$(64),KEY2$(64)
110 KEY1$="LJ;  K+*O PU I-=V C  BXZ4 36 521, .
N M/ R EY TWQ9 07 8<>FHD  GSA"
120 KEY2$="LJ:  K\^O PU I_|V C  BXZ$ #& % ![ ]
N M? R EY TWQ( )' @  FHD  GSA"
130 KS=PEEK(141)*256+PEEK(140)
140 POKE KS+94,34
150 DIM DIR$(20),Q$(54),A$(25),B$(54),F$(8),S(
100),B(100),G$(25),N$(3)
160 DIM NM1$(10),NM2$(10),NM3$(10),NM4$(10),NM
$(40),P(4)
170 FOR L=1 TO 54:B$(L,L)=" ":NEXT L:Q$=B$:A$=
B$(1,25):G$=A$:NM$=B$(1,40)
180 GRAPHICS 0:TRAP 7010
190 POKE 16,64:POKE 53774,64
1000 REM *** FILE CHOICE MENU
1010 POKE 752,1:PRINT "{":POSITION 12,2:PRINT
"TRIVIA GAME"
1015 POSITION 12,4:PRINT "FILE CHOICES":GOSUB
6110:IF F>9 THEN F=9
1020 IF F=0 THEN 1200
1030 OPEN #4,6,0,"D:*.TDB":FOR L=1 TO F:INPUT
#4,DIR$:POSITION 12,L+5:PRINT L;". ";DIR$(3,10
):NEXT L:CLOSE #4
1035 POSITION 12,F+7:PRINT "0. EXIT"
1040 POSITION 12,F+9:PRINT "PRESS (0-";F;")"
1050 GOSUB 6010:IF K2<48 OR K2>F+48 THEN 1050
1055 IF K2=48 THEN 8010
1060 OPEN #5,6,0,"D:*.TDB"
1070 FOR L=1 TO K2-48:INPUT #5,DIR$:NEXT L:CLO
SE #5:F$=DIR$(3,11)
1080 IF F$(LEN(F$),LEN(F$))=" " THEN F$=F$(1,L
EN(F$)-1):GOTO 1080
1090 GOSUB 6710:OPEN #5,4,0,DIR$:INPUT #5;N$:C
LOSE #5:N=VAL(N$):IF N<20 THEN 1250
1100 GOTO 2010
1200 POSITION 8,6:PRINT "NO DATA FILES ON DISK
":POSITION 5,9:PRINT "YOU MUST SWITCH DATA DIS
KS"
1210 POSITION 8,10:PRINT "AND TYPE 'RUN' AGAIN
":POSITION 17,12:PRINT "OR"
1220 POSITION 12,14:PRINT "LOAD AND RUN":POSIT
ION 8,15:PRINT "THE DATA BASE PROGRAM"
1230 POSITION 12,19:PRINT "PRESS ANY KEY":GOSU
B 6010:GOTO 8010
1250 PRINT "{":POSITION 5,8:PRINT "TOO FEW QUE
STIONS IN THE FILE":POSITION 12,9:PRINT "TO PL
AY A GAME"
1260 POSITION 12,11:PRINT "PRESS ANY KEY":GOSU
B 6010:GOTO 1010
```

```
2000 REM *** NAME ENTRY
2010 POKE 752,1:PRINT "{CLEAR}"
2020 POSITION 13,2:PRINT "TRIVIA GAME"
2030 POSITION 4,5:PRINT "HOW MANY PLAYERS?  PR
ESS 1 TO 4"
2040 GOSUB 6010:P=K2-48:IF P<1 OR P>4 THEN 204
0
2050 POSITION 2,20:PRINT "NAMES MAY ONLY BE 10
 CHARACTERS LONG":POKE 752,0
2060 TRAP 2060:POSITION 3,8:PRINT "ENTER PLAYE
R 1'S NAME ";:INPUT NM1$:IF NM1$="" THEN 2060
2070 IF P=1 THEN 2130
2080 TRAP 2080:POSITION 3,10:PRINT "ENTER PLAY
ER 2'S NAME ";:INPUT NM2$:IF NM2$="" THEN 2080
2090 IF P=2 THEN 2130
2100 TRAP 2100:POSITION 3,12:PRINT "ENTER PLAY
ER 3'S NAME ";:INPUT NM3$:IF NM3$="" THEN 2100
2110 IF P=3 THEN 2130
2120 TRAP 2120:POSITION 3,14:PRINT "ENTER PLAY
ER 4'S NAME ";:INPUT NM4$:IF NM4$="" THEN 2120
2130 TRAP 7010:POKE 752,1:NM$(1,10)=NM1$:NM$(1
1,20)=NM2$:NM$(21,30)=NM3$:NM$(31,40)=NM4$
3000 REM *** GAME
3010 POKE 752,1:PRINT "{CLEAR}"
3020 POSITION 11,8:PRINT ".... PLEASE WAIT":PO
SITION 12,11:PRINT "INDEXING DATA"
3030 OPEN #4,12,0,DIR$:INPUT #4,N$:N=VAL(N$):F
OR L=1 TO N:NOTE #4,X,Y:S(L)=X:B(L)=Y:INPUT #4
,Q$,A$:NEXT L
3040 POSITION 10,14:PRINT "ORDERING QUESTIONS"
3050 FOR L=1 TO N:S(0)=S(L):B(0)=B(L):R=INT(RN
D(1)*N)+1:S(L)=S(R):B(L)=B(R):S(R)=S(0):B(R)=B
(0):NEXT L
3060 PRINT "{CLEAR}":GOSUB 6210:M=0:P(1)=0:P(2)=0:P(
3)=0:P(4)=0
3070 FOR L=1 TO P*5:GOSUB 6410
3075 FL=1:IF RND(1)<0.1 THEN FL=2:POSITION 10,
15:PRINT "BONUS POINT QUESTION"
3080 M=M+1:IF M=P+1 THEN M=1
3090 POSITION 3,4:PRINT "GET READY ";NM$(M*10-
9,M*10):FOR D=1 TO 200:NEXT D
3100 POSITION 3,4:PRINT "GET SET ";NM$(M*10-9,
M*10);"  ":FOR D=1 TO 200:NEXT D
3110 POSITION 3,4:PRINT "GO ";NM$(M*10-9,M*10)
;"        "
3120 POINT #4,S(L),B(L):INPUT #4,Q$,A$:GOSUB 6
810
3130 T=0:Z=500:POKE 752,0:POSITION 10,12:PRINT
  " ";
3140 GOSUB 6610:IF Z=0 THEN 3200
```

```
3150 IF K=12 THEN 3200
3160 IF K=52 AND T>0 THEN POSITION T+10,12:PRI
NT " ←";:G$(T,T)=" ":T=T-1:GOTO 3140
3170 IF T=25 THEN 3140
3180 IF K2=32 AND K<>33 THEN 3140
3190 T=T+1:G$(T,T)=CHR$(K2):POSITION T+10,12:P
RINT CHR$(K2);:GOTO 3140
3200 POKE 752,1:IF G$<>A$ THEN 3220
3210 POSITION 7,15:PRINT "CORRECT - ";INT(Z*FL
/20);" POINTS SCORED":P(M)=P(M)+INT(Z*FL/20):G
OSUB 6410:GOTO 3300
3220 POSITION 10,15:PRINT " INCORRECT ANSWER
 "
3300 FOR D=1 TO 200:NEXT D:POSITION 3,15:PRINT
 B$(1,34):G$=B$(1,25):GOSUB 6510:POSITION 33,4
:PRINT 25
3310 NEXT L
3320 POSITION 6,15:PRINT "GAME OVER --- PRESS
ANY KEY":POKE 764,255:GOSUB 6010:CLOSE #4
3330 PRINT "{CLEAR}":POSITION 14,5:PRINT "RANKING"
3340 FOR L=1 TO P:POSITION 10,L+6:M=1:FOR L2=1
 TO P:IF P(L2)>P(M) THEN M=L2
3350 NEXT L2:PRINT NM$(M*10-9,M*10);"    ";P(M)
:P(M)=-1:NEXT L
3360 POSITION 12,P+8:PRINT "PRESS ANY KEY":POK
E 764,255:GOSUB 6010
3370 PRINT "{CLEAR}":POSITION 5,5:PRINT "PLAY AGAIN?
  PRESS Y OR N"
3380 GOSUB 6010:IF K2=78 THEN 8010
3390 IF K2<>89 THEN 3380
3400 POSITION 2,8:PRINT "SAME PLAYERS AND FILE
?  PRESS Y OR N"
3410 GOSUB 6010:IF K2=78 THEN RUN
3420 IF K2<>89 THEN 3410
3430 OPEN #4,12,0,DIR$:GOTO 3040
6000 REM *** KEY INPUT ROUTINE
6010 K=PEEK(764):IF K>127 THEN 6010
6020 POKE 764,255:K2=PEEK(K+KS):RETURN
6100 REM *** FILE COUNTING ROUTINE
6110 TRAP 6150:F=0
6120 OPEN #4,6,0,"D:*.*"
6130 INPUT #4,DIR$:IF DIR$(11,13)<>"TDB" THEN
6130
6140 F=F+1:GOTO 6130
6150 CLOSE #4:IF PEEK(195)<>136 THEN 7010
6160 TRAP 7010:RETURN
6200 REM *** SCREEN PRINTING ROUTINE
6210 PRINT "{CLEAR}":POKE 752,1:POSITION 1,1:PRINT "
{INVERSE}";:POSITION 1,22:PRINT "{INVERSE}";
```

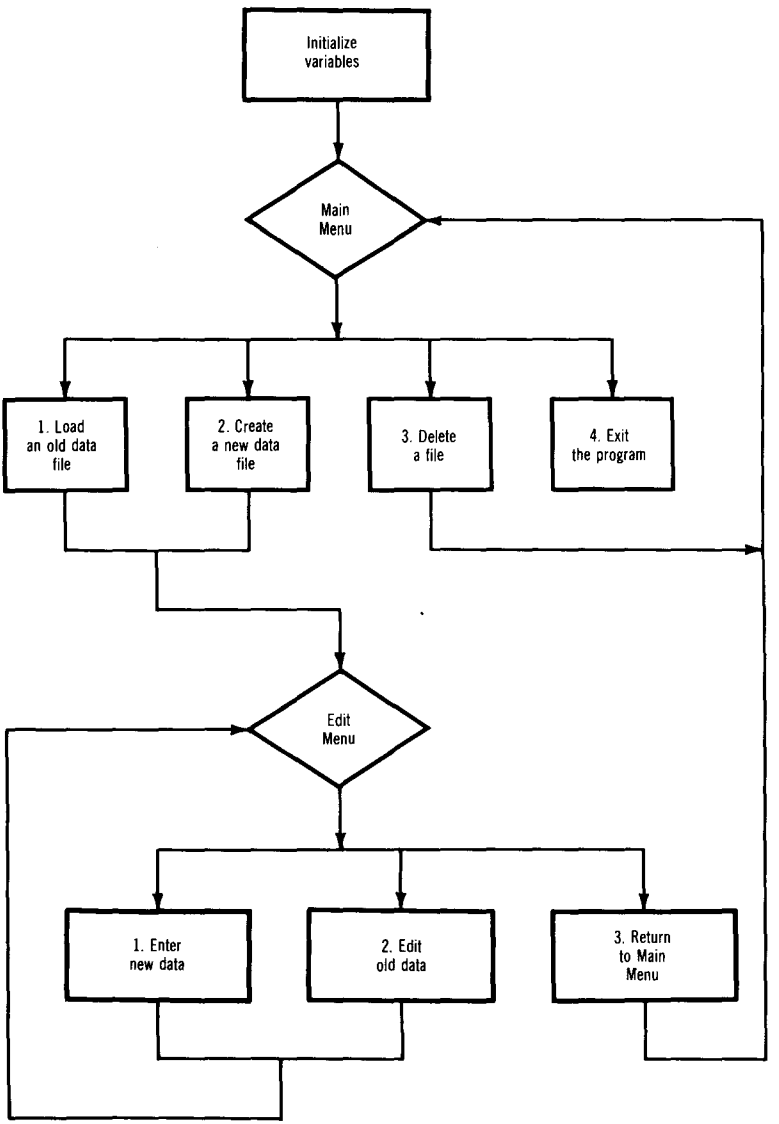
```
6220 FOR C=2 TO 37:POSITION C,1:PRINT "─";:POS
ITION C,22:PRINT "─";:NEXT C
6230 POSITION 38,1:PRINT "┐";:POSITION 38,22:P
RINT "┘";
6240 FOR R=2 TO 21:POSITION 1,R:PRINT CHR$(124
);:POSITION 38,R:PRINT CHR$(124);:NEXT R
6250 POSITION 15,2:PRINT "TRIVIA GAME"
6260 POSITION 1,3:PRINT "├";:FOR C=2 TO 37:PRI
NT "─";:NEXT C:PRINT "┤"
6270 POSITION 25,4:PRINT "TIMER:  25"
6280 POSITION 1,5:PRINT "├";:FOR C=2 TO 37:PRI
NT "─";:NEXT C:PRINT "┤"
6290 POSITION 3,7:PRINT "QUESTION:"
6300 POSITION 14,8:FOR C=1 TO 22:PRINT "─";:NE
XT C
6310 POSITION 4,10:FOR C=1 TO 32:PRINT "─";:NE
XT C
6320 POSITION 1,14:PRINT "├";:FOR C=2 TO 37:PR
INT "─";:NEXT C:PRINT "┤"
6330 POSITION 3,12:PRINT "ANSWER?":POSITION 11
,13:FOR C=1 TO 25:PRINT "─";:NEXT C
6340 POSITION 1,16:PRINT "├";:FOR C=2 TO 37:PR
INT "─";:NEXT C:PRINT "┤"
6350 POSITION 5,17:PRINT "PLAYER'S NAME","SCOR
E"
6360 POSITION 8,18:PRINT NM1$:POSITION 8,19:PR
INT NM2$:POSITION 8,20:PRINT NM3$:POSITION 8,2
1:PRINT NM4$
6370 RETURN
6400 REM *** SCORE PRINTING ROUTINE
6410 POKE 752,1:POSITION 26,18:PRINT P(1):IF P
>1 THEN POSITION 26,19:PRINT P(2)
6420 IF P>2 THEN POSITION 26,20:PRINT P(3):IF
P>3 THEN POSITION 26,21:PRINT P(4)
6430 RETURN
6500 REM *** BLANKING ROUTINE
6510 POKE 752,1:POSITION 14,7:PRINT B$(1,22);:
POSITION 4,9:PRINT B$(1,32);:POSITION 11,12:PR
INT B$(1,25);
6520 RETURN
6600 REM *** TIMER-KEY ROUTINE
6610 K=PEEK(764):Z=Z-1:POKE 752,1:POSITION 33,
4:PRINT INT(Z/20);"  ";:POKE 752,0:POSITION T+
11,12:PRINT " ←";
6620 IF Z=0 THEN 6650
6630 IF K=255 THEN 6610
6640 K2=PEEK(K+KS)
6650 POKE 764,255:RETURN
6700 REM *** FILE NAME ROUTINE
```

```
6710 DIR$(1,2)="D:":DIR$(3,LEN(F$)+3)=F$:DIR$(
LEN(F$)+3,LEN(F$)+7)=".TDB":DIR$(LEN(F$)+7,20)
="                    "
6720 RETURN
6800 REM *** QUESTION PRINT ROUTINE
6810 POKE 752,1:POSITION 14,7:PRINT Q$(1,22);:
POSITION 4,9:PRINT Q$(23,54);:RETURN
7000 REM *** DISK ERROR ROUTINE
7010 TRAP 7050:POKE 752,1:PRINT "{CLEAR}":E=PEEK(195
):POSITION 9,6:PRINT "ERROR ";E;" AT LINE ";PE
EK(186)+PEEK(187)*256
7020 IF E>18 THEN POSITION 13,8:PRINT "DISK ER
ROR"
7030 POSITION 12,11:PRINT "PRESS ANY KEY":POKE
 764,255:GOSUB 6010:CLOSE #4:CLOSE #5
7040 IF E=144 THEN 8010
7050 RUN
8000 REM *** EXIT
8010 PRINT "{CLEAR}":POKE 16,192:POKE 53774,247:POKE
 752,0:END
```

Data Base Flowchart 1-A Overview

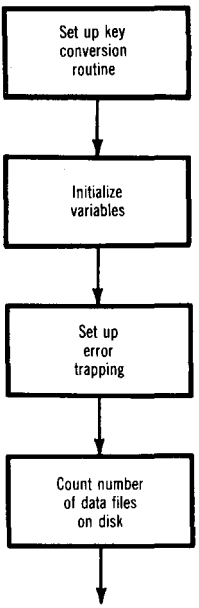

(foldout)

Flowchart 1-A



Data Base Flowchart 1-B Initialize variables

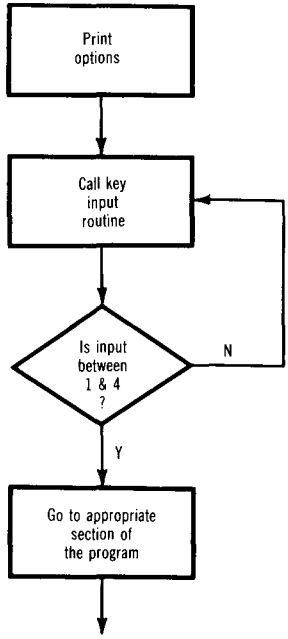
Flowchart 1-B



Data Base Flowchart 1-C Main Menu

 (foldout)

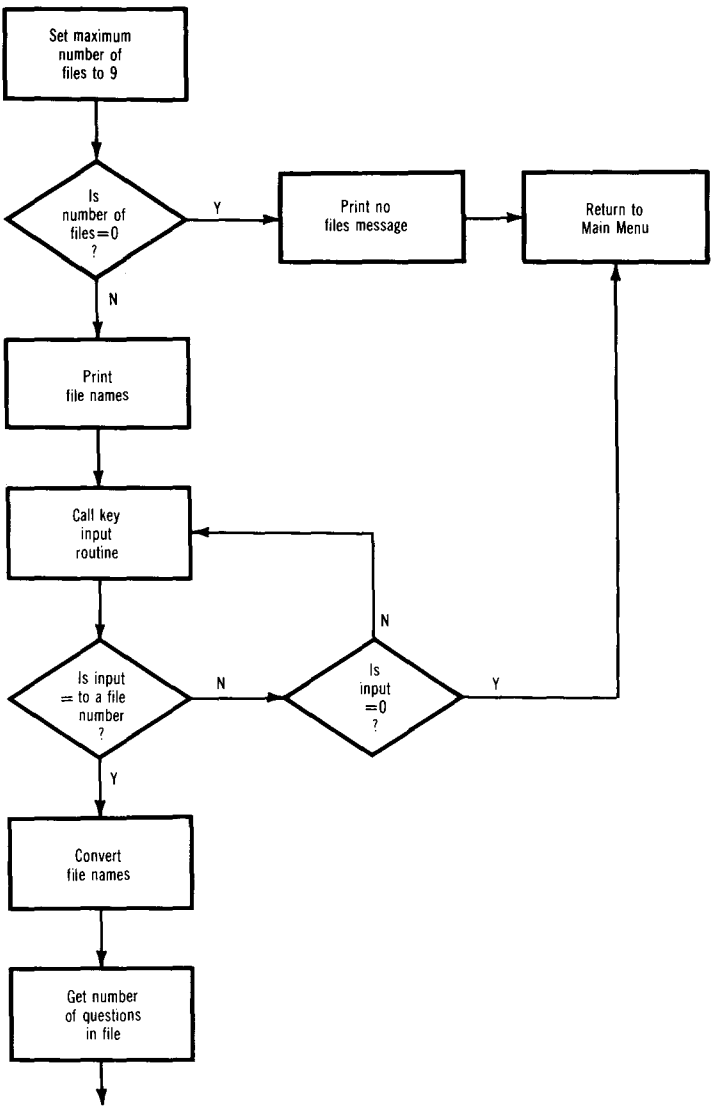
Flowchart 1-C



Data Base Flowchart 1-D Load a file


(foldout)

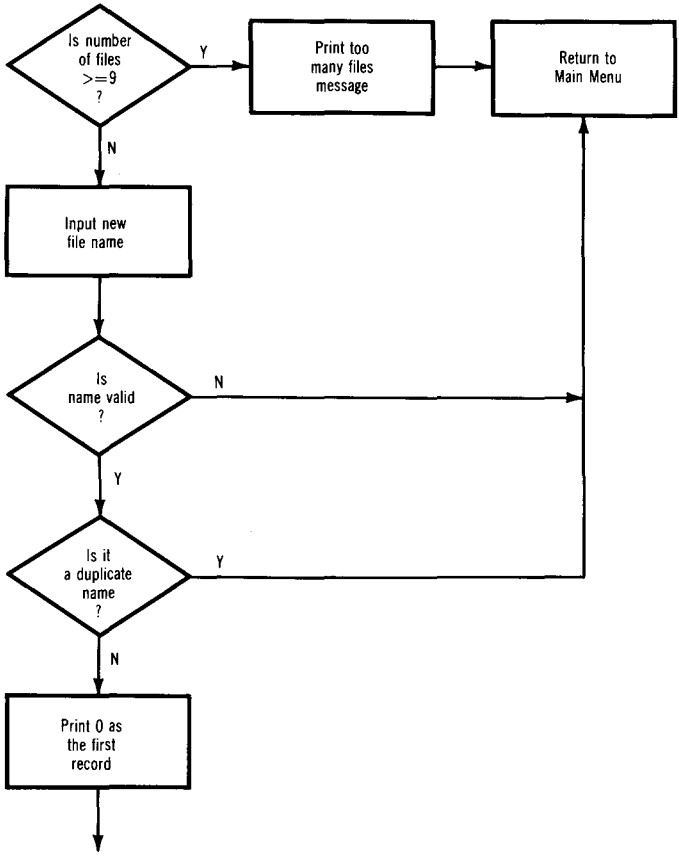
Flowchart 1-D



Data Base Flowchart 1-E Create a file


(foldout)

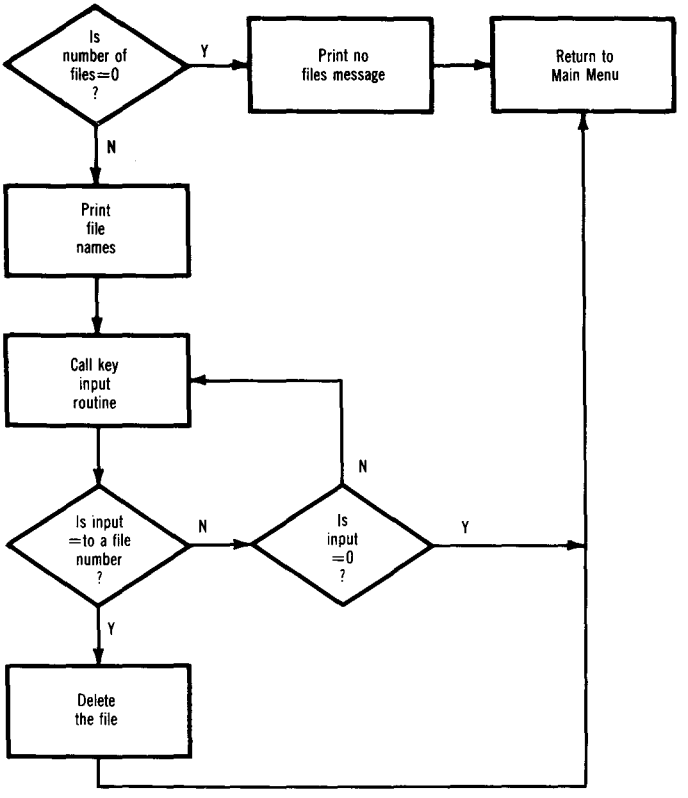
Flowchart 1-E



Data Base Flowchart 1-F Delete a file

(foldout)

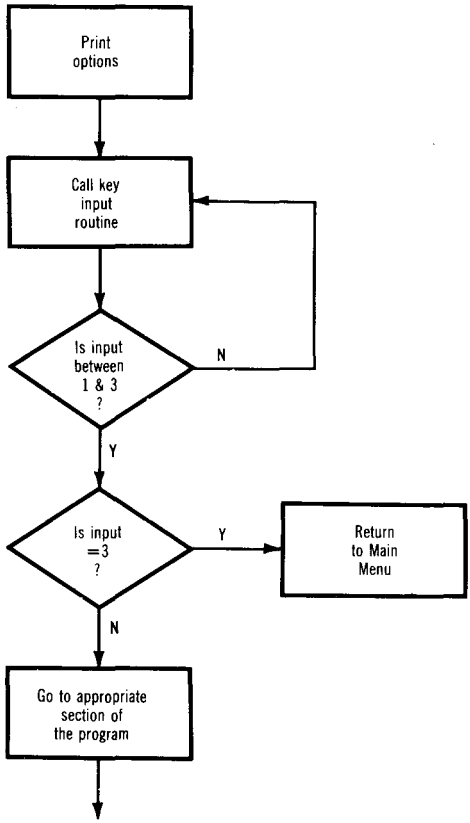
Flowchart 1-F



Data Base Flowchart 1-G Edit Menu


(foldout)

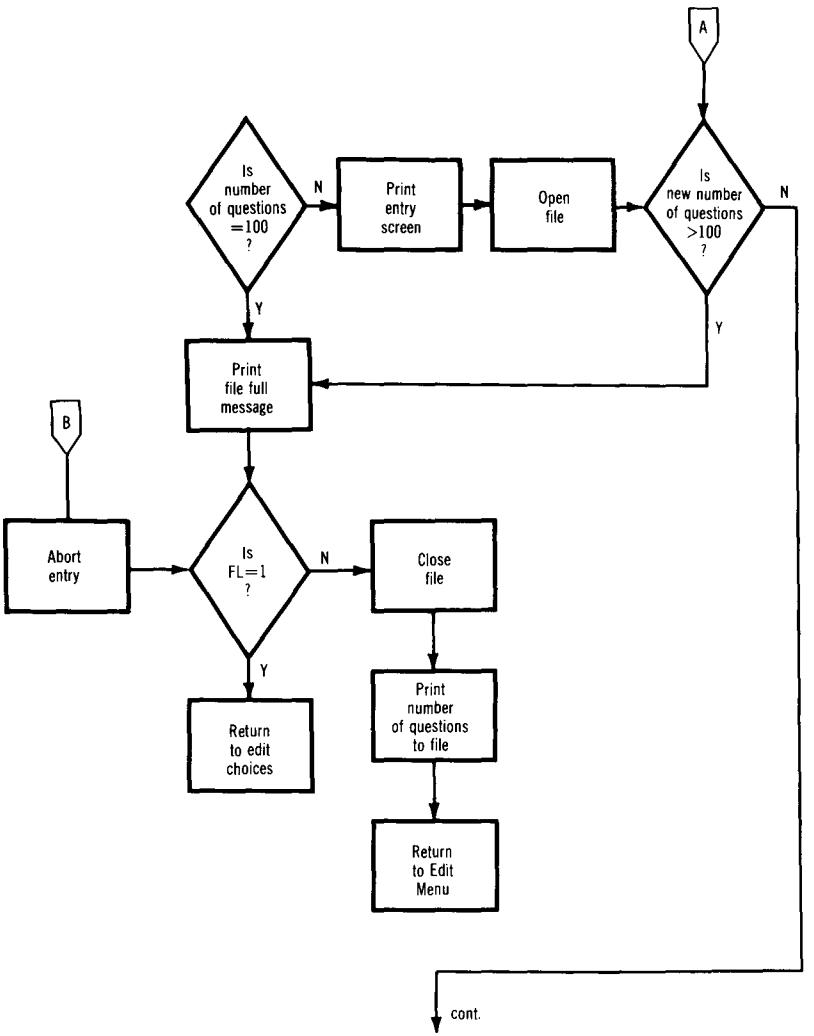
Flowchart 1-G



Data Base Flowchart 1-H Enter data

(foldout)

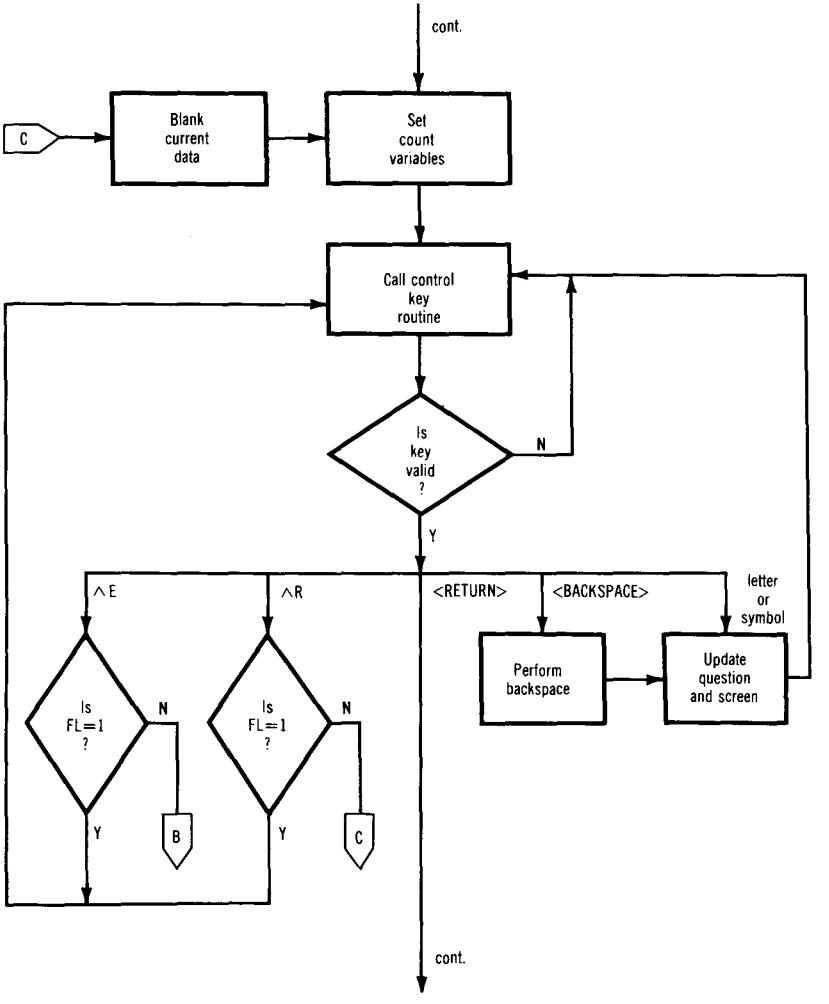
Flowchart 1-H



Data Base Flowchart 1-H cont. Enter data


(foldout)

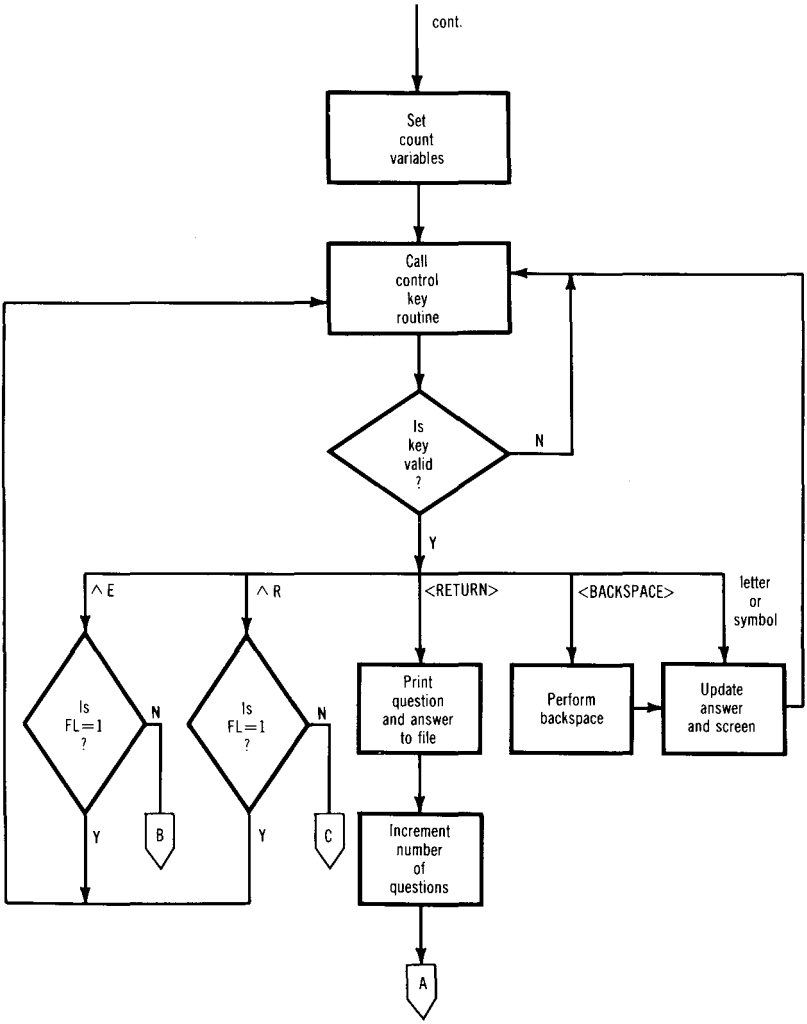
Flowchart 1-H cont.



Data Base Flowchart 1-H cont. Enter data


(foldout)

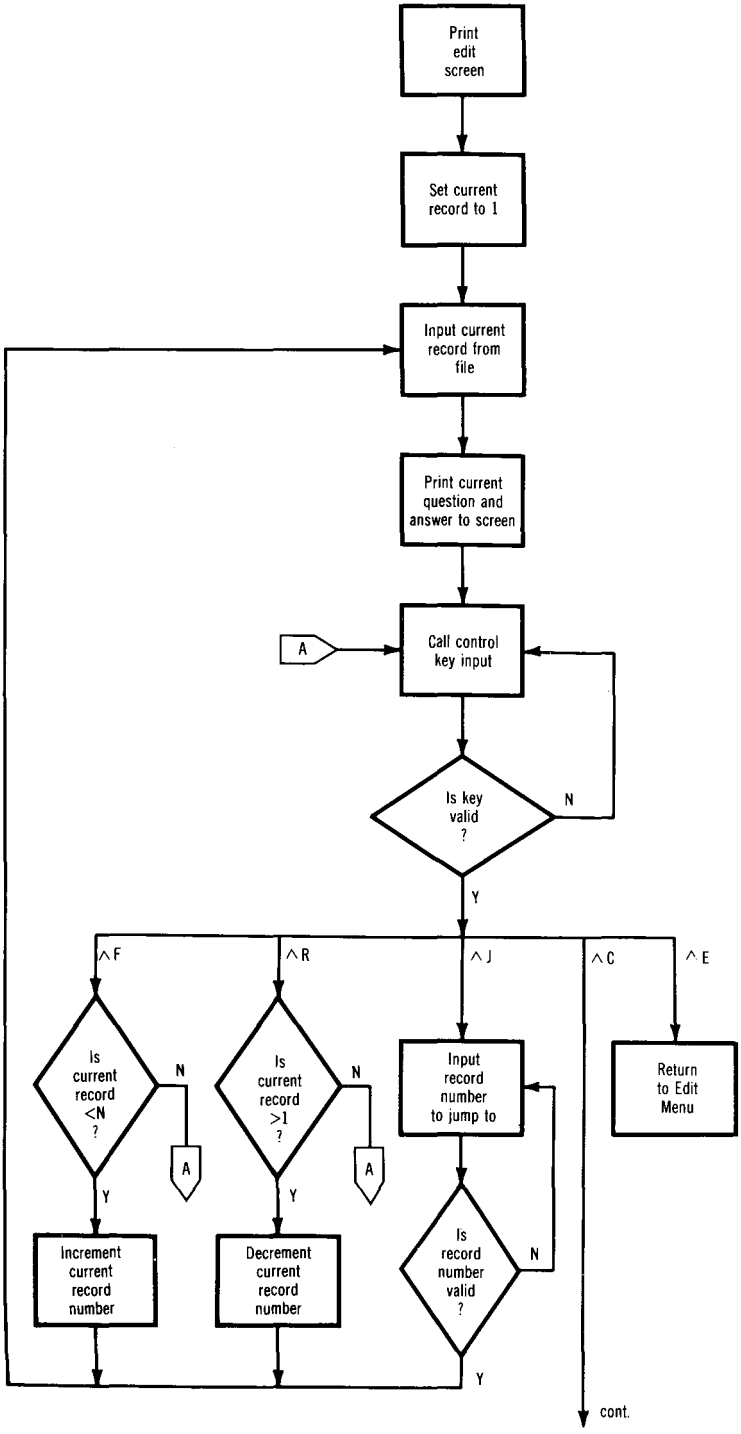
Flowchart 1-H cont.



Data Base Flowchart 1-I Edit data


(foldout)

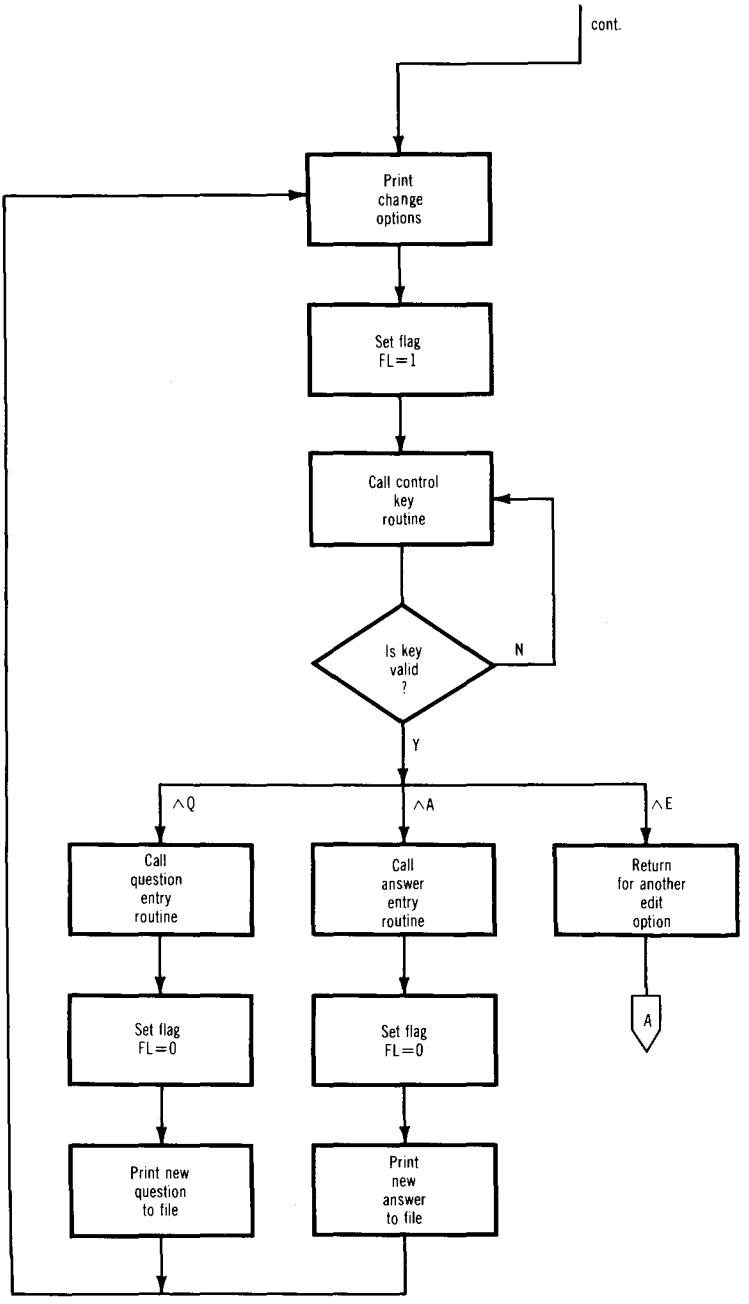
Flowchart 1-I



Data Base Flowchart 1-I cont. Edit data


(foldout)

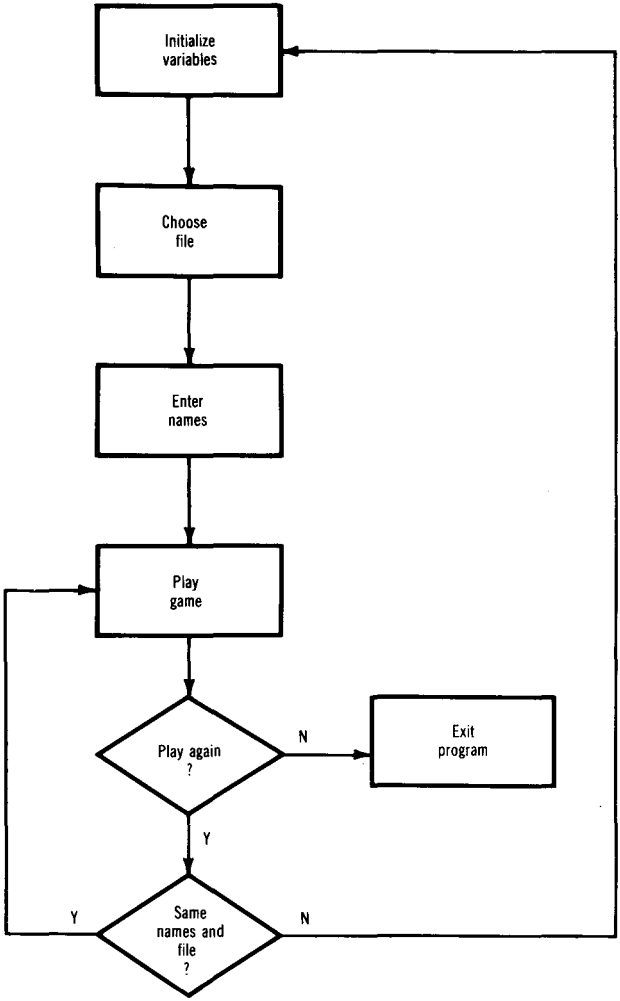
Flowchart 1-I cont.



Game Flowchart 2-A Overview

(foldout)

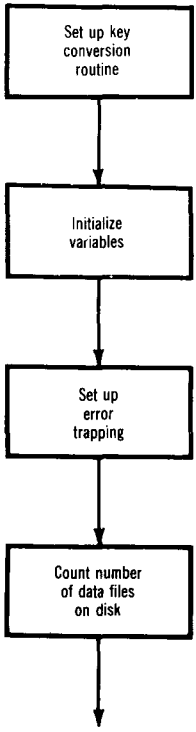
Flowchart 2-A



Game Flowchart 2-B Initialize variables


(foldout)

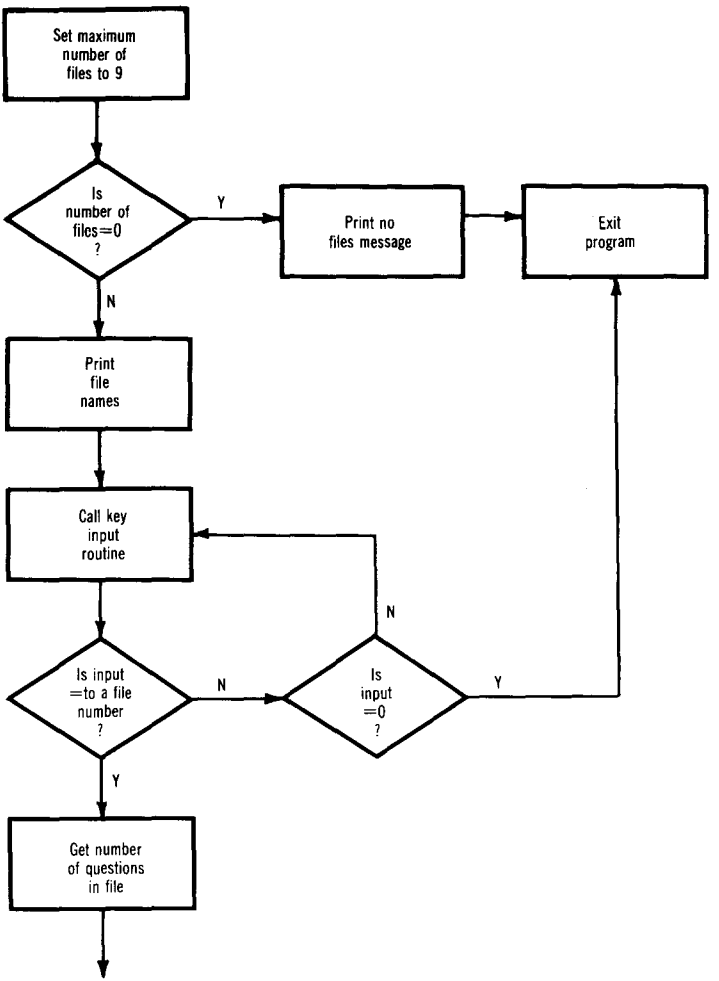
Flowchart 2-B



Game Flowchart 2-C Choose a file


(foldout)

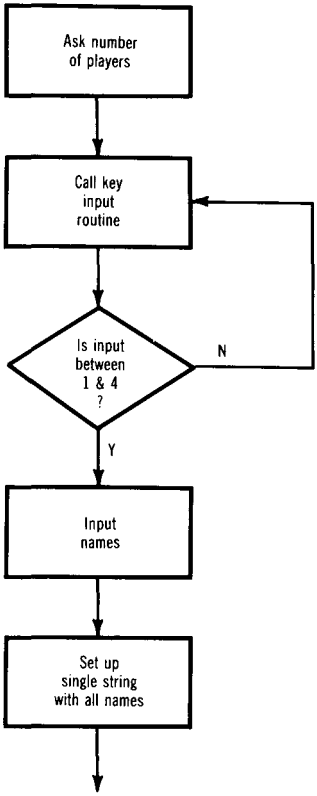
Flowchart 2-C



Game Flowchart 2-D Enter names

(foldout)

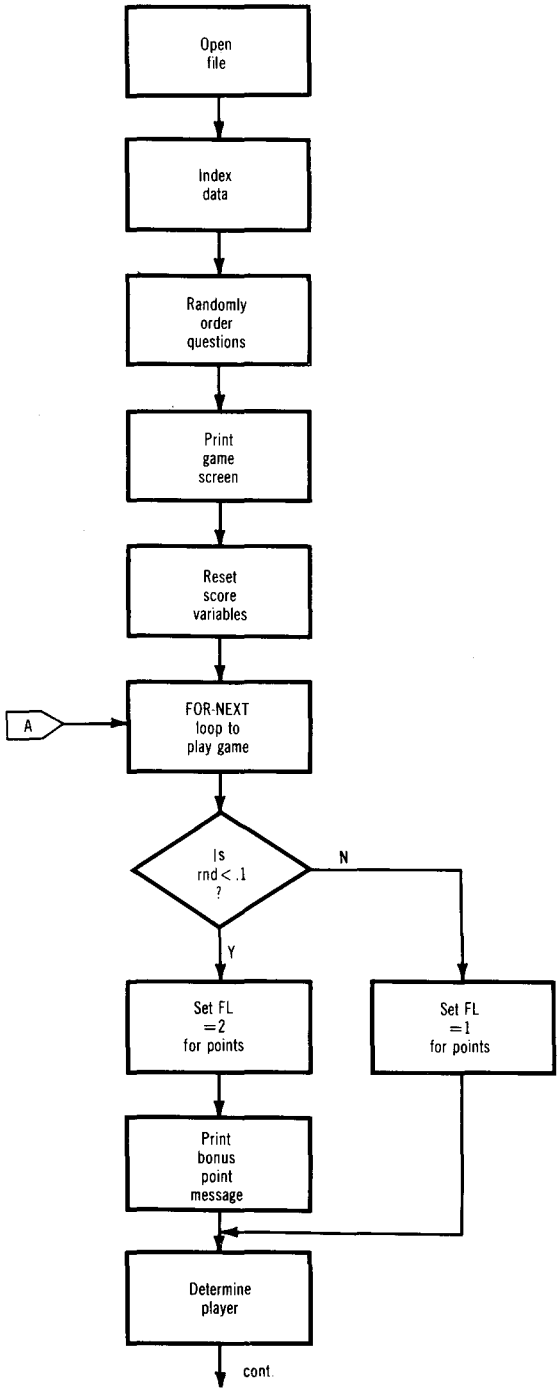
Flowchart 2-D



Game Flowchart 2-E Play the game

(foldout)

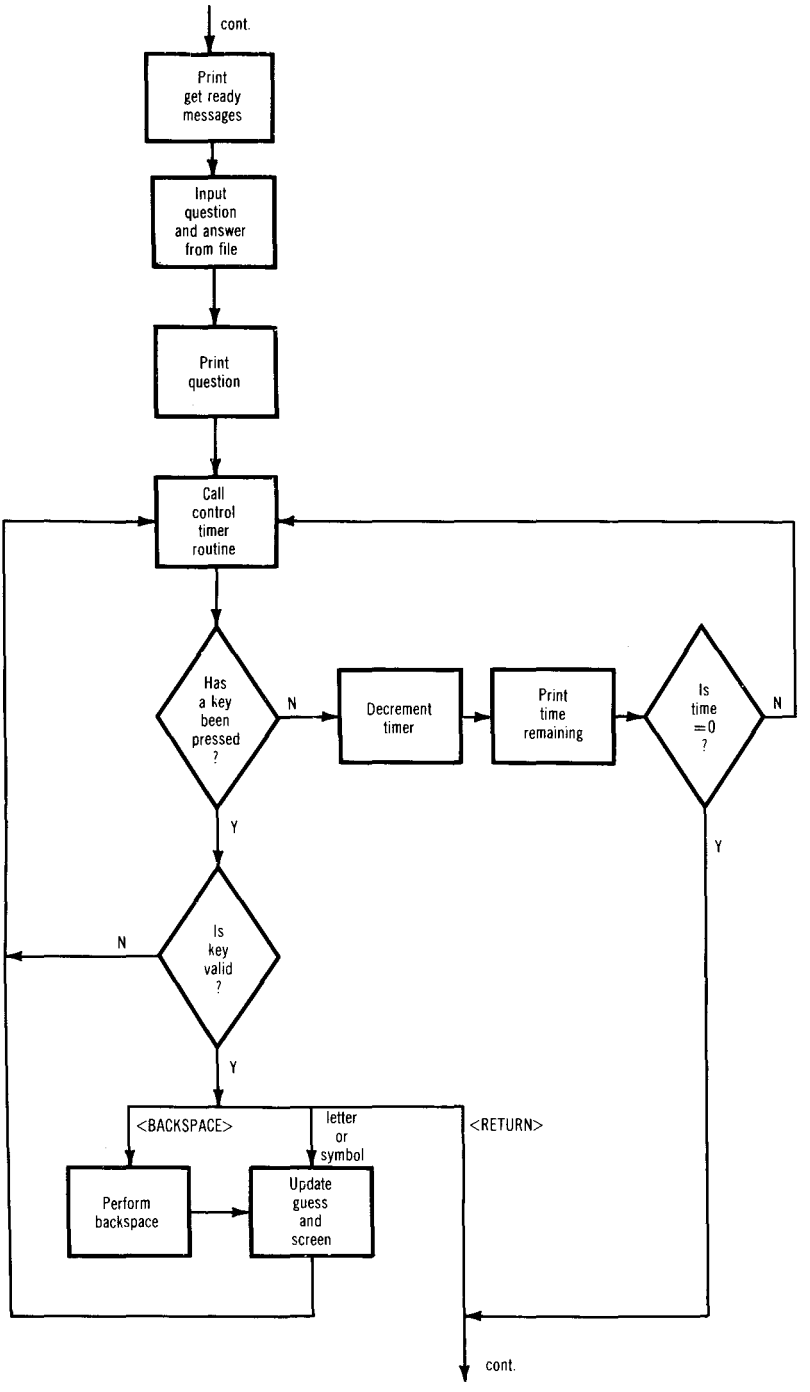
Flowchart 2-E



Game Flowchart 2-E cont. Play the game


(foldout)

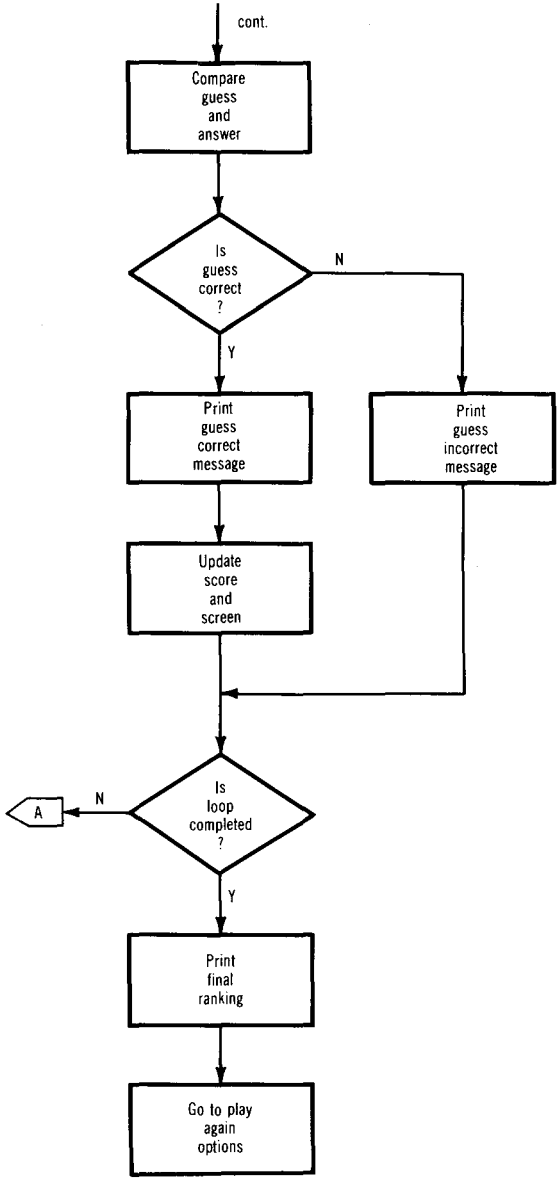
Flowchart 2-E cont.



Game Flowchart 2-E cont. Play the game


(foldout)

Flowchart 2-E cont.



SAMS

ATARI® Trivia Data Base

- Includes listings of two complete programs, a data base and a trivia game
- Introduces ATARI owners to data bases and their applications by enabling users to understand how a simplified data base is created and why it works
- Discusses user-friendly data entry, error checking and program continuity
- Shows how to design file records

ATARI Trivia Data Base is an entertaining way to see for yourself how data bases can be used for fun and education, as well as for business

Machine Requirements:

- **Atari 800XL or Atari 800 (64K)**
- **1 Disk Drive (DOS 2.0)**

Howard W. Sams & Co., Inc.
A Publishing Subsidiary of ITT
4300 West 62nd Street, Indianapolis, Indiana 46268 U.S.A.

$8.95/22397

ISBN: 0-672-22397-X