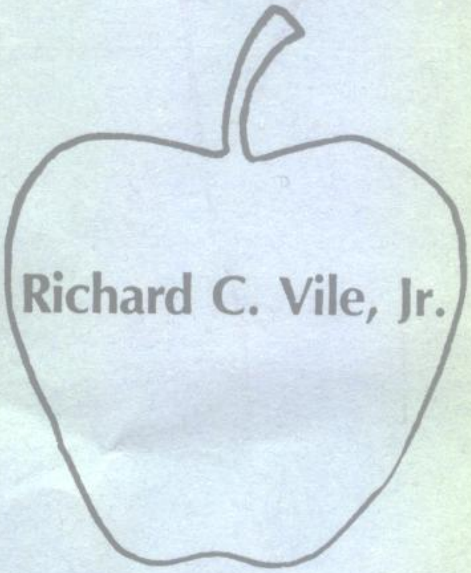
apple II
PROGRAMMER'S HANDBOOK

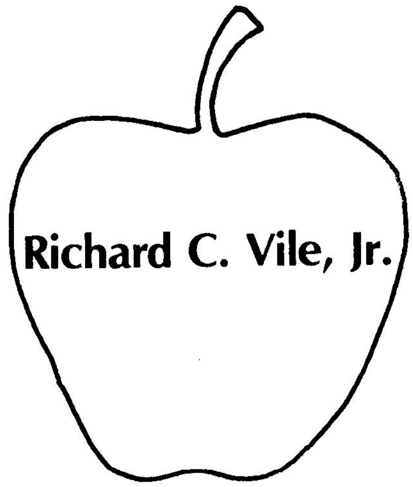


Richard C. Vile, Jr.

PROGRAMS AND PROGRAMMING TECHNIQUES
IN APPLE INTEGER BASIC, APPLESOFT BASIC,
PASCAL, AND 6502 ASSEMBLY LANGUAGE

apple II

PROGRAMMER'S HANDBOOK



Richard C. Vile, Jr.

PROGRAMS AND PROGRAMMING TECHNIQUES
IN APPLE INTEGER BASIC, APPLESOFT BASIC,
PASCAL, AND 6502 ASSEMBLY LANGUAGE

GRANADA
London Toronto Sydney New York

Granada Publishing Limited—Technical Books Division
Frogmore, St. Albans, Herts AL2 2NF
and
36 Golden Square, London W1R 4AH
515 Madison Avenue, New York, NY 10022, USA
117 York Street, Sydney, New South Wales 2000, Australia
100 Skyway Avenue, Rexdale, Ontario, Canada M9W 3A6
61 Beach Road, Auckland, New Zealand

ISBN 0-246-12027-4

First published in the United States of America 1982 by
Prentice-Hall, Inc.
First published in Great Britain 1983 by Granada Publishing Ltd.

Printed in the United States of America

RICHARD C. VILE, JR. is a senior software specialist for GemNet Software Corporation in Ann Arbor, Michigan. Active in the computer field since 1974, Mr. Vile has taught mathematics and computer science at Eastern Michigan University, worked as a programmer, and written articles for several leading computer magazines.

Preface: How to Use This Book

This is a book *of* and *about* software. It emphasizes the use of four programming languages available on the APPLE II personal computer:

- Integer BASIC
- APPLESOFT BASIC
- APPLE (UCSD) Pascal
- APPLE (6502) Assembler

It is a book *of* software since there are over 40 programs whose source text have been printed in their entirety in the book. These are *complete* programs, tested and ready to use—such as you might buy in your local computer store. They include such applications as:

- Graphics

  Video—an animated TV test pattern.
  Alphabetics—a collection of entertaining and educational programs for kids.
  Dazzle—dynamic abstract low resolution graphics.

- Education

  Duelling DO Loops—interactive fun with BASIC FOR statements.
  APPLE Trivia—several versions of a configurable quiz program.

- Utilities

  BASIC subroutine loader.
  Pascal Units—calendar, low-res graphics, etc.
  An APPLESOFT Text Editor.

- Languages

  Hex calculator interpreter.
  Scanning programs.

- Entertainment

  The Fifteen Puzzle—interactive game and intelligent program versions.
  The Analogies Generator—nonsense phrases from the computer.

Some programs appear in more than one version. This allows comparison of the strengths and weaknesses of the languages in use. All of the programs may be modified and this is in fact encouraged. That's one of the reasons why we say the book is *about* software. In addition to the programs themselves, there is ample commentary concerning the features and design of many of the programs. There is discussion about the language constructs used in the programs. Plus there are over 50 examples of what we have chosen to call APPLE Programming Tips.

A programmer is one who programs

Each programming tip is highlighted in the text by a banner like the one above and a brief descriptive title, such as "A programmer is one who programs." This saying exemplifies the philosophy of the book. We believe that in order to learn about software or programming, you must engage in programming yourself.

Read other peoples' programs

One excellent way to learn about programming is to read other peoples' programs. That is why we have included complete and substantial programs. To get the most out of the book, you should *read and study* the programs, as well as run them.

That is not to say that we don't believe in the *mechanics* of programming. This is not our aim in this particular book. We will assume that you have learned or are learning about the languages from one of the many excellent introductory texts on language rules and regulations that are available. We hope to take you a step beyond those tutorials, however, and show you how programs are constructed.

Organize your programs for understanding

Since this book is for relative beginners to the art of programming, yet wishes to present substantial examples of real programs, the programs included here have been written with the conscious assumption that people will *read* them. Consequently, the keyword in their construction has been *clarity*. Techniques to make the programs *more efficient* have been avoided in most cases, since that would have made them harder to understand. We hope that most programs in this book can be read and understood using a top-down approach. That is, by first absorbing the higher level structure and then gradually working down into the details. Most of the programs presented use many subroutines or procedures which make this process easier.

The book has been divided into four sections corresponding to the four languages mentioned above. There is some cross referencing of programs. For example, the Amper-Letters program of Chapter 11 uses some assembly language routines from Chapter 19. The APPLE Trivia Quiz program is reincarnated in APPLESOFT and Pascal after being introduced in Integer BASIC.

Explore!

At the end of most Chapters there is a section entitled "Explorations." In these sections, we present suggestions for actual programming. These range from the trivial to the extremely difficult. They give you your chance to practice what we have preached throughout the book. They are called Explorations because of their somewhat open-ended nature. We hope that they will lead you to experiment with ideas and modifications of your own. Experimenting with new ideas, modifying programs step by step, trying out new techniques: all these activities will lead you to a better understanding of programming and to more enjoyment of your APPLE.

Happy programming and enjoy!!

Dick Vile

Contents

18/1-37

Chapter 1

APPLE Integer BASIC

In this chapter, we give a very brief summary of Integer BASIC, indicating some of its desirable features and their usefulness. This is not intended to supplant either APPLE's tutorial manual on the subject or any other tutorial materials. We assume that the reader is familiar with some dialect of BASIC, probably Integer BASIC, and do not pretend that this is a beginner's guide. Many of the points we make in this section will be amplified upon in later chapters.

INTRODUCING INTEGER

Integer BASIC was the first APPLE language after 6502 machine language. Folklore has it that Steve Wozniak implemented the entire interpreter using only the mini-assembler of the original Monitor ROM as a tool—truly an impressive feat.

Integer BASIC is a somewhat stripped down version of the BASIC language: it does not support floating-point variables or functions, has limited string capabilities, provides no user-defined functions, and lacks multidimensional arrays. Nonetheless it is a useable language and provides some features not commonly found in many implementations of BASIC: arbitrary length identifiers, support for (low-resolution) color graphics, access to game paddle inputs, memory-mapped display support, the ability to CALL machine language routines, etc.

Playing Games

Integer BASIC provides enough features to allow interesting games to be written. In particular it allows color graphics displays to be produced in sixteen different colors on the low-resolution graphics display. The programmer is given the ability to plot points, draw horizontal and vertical lines, and to change colors. Of course, there are statements to allow changing from text to graphics modes and vice-versa.

Integer BASIC Graphics Commands

```
PLOT COL,ROW
HLIN COL1,COL2 AT ROW1
VLIN ROW1,ROW2 AT COL1
COLOR = N
GR
TEXT
```

These commands are pretty simple, yet comprehensive enough to allow fairly sophisticated graphics to be produced. Some of these capabilities are demonstrated in Chapters 3, 4, and 6.

In addition to the color graphics capabilities, Integer BASIC programs have access to two or more game paddle inputs via the built-in function PDL.

PDL(n)

returns the current reading of the *n*th game paddle, where *n* is between 0 and 3. The standard APPLE comes equipped with two paddles corresponding to *n* = 0 and *n* = 1. Most Integer BASIC programs make do with these two, although it is possible to add two more paddles to the APPLE.

The APPLE II has a built-in speaker. This is not supported directly by a statement like PDL, but it is possible to access the speaker using the PEEK or POKE functions. The memory location numbered −16336 is "tied" to the speaker in such a way that every time it is referenced as the argument of a PEEK or a POKE, the speaker makes a tiny click. Playing around with various combinations of such references in loops, with delays in between will produce various noises and tones. This is illustrated by the following brief program. Try running it and twisting the PDL(1) control back and forth very slowly. With some experimentation, you will see why it has been dubbed the "squeaky-door" demo. See Listing 1.1 at the end of this chapter.

Display Features

The APPLE II computer provides a memory-mapped display. This means that an area of memory is dedicated to storing the information that appears on the screen. Whenever one of these memory locations is changed, the difference is instantaneously visible on the screen. Integer BASIC takes some advantage of this design by allowing the cursor on the screen to be positioned to any desired row and column before a PRINT statement is issued. This allows a program to selectively update portions of the screen without destroying other information on the remainder of the screen. The cursor positioning statements are:

VTAB row
TAB col

These are taken full advantage of in the APPLE Trivia quiz of Chapter 2, as well as in many other programs.

In addition to the VTAB and TAB commands, Integer BASIC programs can take advantage of other features of the display. In particular, the Monitor ROM routines are used by the interpreter to do screen output. Certain parameters which these routines make use of may be directly controlled by the use of POKE statements. In particular, the concept of a scrolling window is frequently made use of in Integer BASIC programs. We deal with this concept and its associated parameters in Chapter 2 as well.

Coding Advantages of Integer BASIC

Integer BASIC offers some significant advantages over many other versions of BASIC in writing clear and easily understood programs. Probably the biggest of these advantages is its rules regarding identifiers.

1. Integer BASIC variable names may be as long or as short as the programmer wishes.
2. Variable names may be used in place of line numbers in GOTO and GOSUB statements.

The use of long variable names which may be chosen to directly suggest the intended use of the variable in the program has significant psychological advantages when reading programs. Instead of having to remember, for example, that variable S1 represents a score, the programmer may simply use SCORE as the name of the variable.

The ability to give names to subroutines which the program calls is another significant advantage both in reading and understanding programs. Compare, for example:

```
100 GOSUB 1000          100 GOSUB
                            INITIALIZE
105 GOSUB 9000          105 GOSUB
                            INTRODUCTION
110 GOSUB 5000          110 GOSUB
                  with      PLAYGAME
120 GOSUB 8000          120 GOSUB
                            ASKREPEAT
125 IF A$="YES"         125 IF ANSWER$=
      THEN 110              "YES" THEN 110
130 END                 130 END
```

To use Integer BASIC most effectively, a programmer should take full advantage of these capabilities, choosing names which are suggestive and which will lead to a better understanding for those who will read the programmer's code—that includes the programmer her/himself as well. We try very hard to practice what we preach in all the Integer BASIC programs in this book.

LISTINGS

LISTING 1.1 SQUEAKY DOOR DEMO

```
>LIST
  5 SPKR=-16336
 10 X= PEEK (SPKR)+ PEEK (SPKR)
 15 REM ============================
 16 REM = VARY THE FOLLOWING DELAY =
 17 REM = LOOP TO PRODUCE THE      =
 18 REM = SQUEAKY DOOR EFFECT.     =
 19 REM ============================
 20 FOR I=1 TO PDL (1): NEXT I
 30 GOTO 10

>
```

Chapter

2

Interactive Programs: Using Menus

A large number of APPLE II programs take the form of interactive dialogues with the user. A standard approach to the user interface is the use of *menus*. In this chapter we discuss techniques for effective, user-friendly interactive programs. In the process, we present an example of a menu-driven, interactive program, which illustrates many of the techniques discussed.

1. EFFECTIVE USE OF THE APPLE II SCREEN

The APPLE II features a *memory-mapped* display. This means that a portion of the APPLE's system memory is set aside to hold the information which appears on the screen. When a new value is stored into one of these locations, the difference is *instantaneously* visible on the screen. As a consequence, portions of the display may be selectively updated without affecting the rest.

Cursor Controls

The APPLE's text display holds 24 lines of 40 characters each. A BASIC program displays text on the screen by the use of PRINT statements. The output goes to the screen at the "current" row and column position. This position is remembered by BASIC as the program runs, and may be *modified* by the use of the VTAB and TAB statements.

VTAB This statement positions the screen cursor at the row whose number is the value of the expression following VTAB:

```
VTAB 23
VTAB I + 5
VTAB 2*J + 1
```

TAB This positions the screen cursor at the column whose number is the value of the expression following TAB:

```
TAB 39
TAB 3*I + 2
```

The VTAB and TAB statements operate independently of each other. In particular, the following sequence of statements:

```
TAB 20
VTAB 15
TAB 10
```

will leave the cursor on row 15 and column 10. In some systems, the fact that the cursor started at column 20 when the TAB 10 statement was executed would cause the row position to advance to 16. This is *not* the case in APPLE Integer BASIC.

The limits on TAB and VTAB are:

TAB:	1–40
VTAB:	1–24

Any attempts to go beyond these limits will earn you a severe reprimand from the Integer BASIC Interpreter. The production of pleasing and effective displays will rely heavily on the use of TAB and VTAB.

Monitor ROM Support Routines

Routines in the APPLE Monitor ROM or Autostart ROM control the flow of information to the screen. We will make use of several of these on a regular basis—mainly to erase parts of the screen. ROM routines are accessed via the CALL statement:

```
CALL constant   or   CALL IDENTIFIER
```

For example, CALL -936 will cause the text screen to be erased and the cursor to return to the upper left-hand corner of the screen. The routines we shall rely on most are summarized in Table 2.1. These routines can be quite helpful in maintaining a neat screen appearance.

Highlighting—Use of Inverse Mode

The use of inverse video can also be quite effective in highlighting or emphasizing various portions of a screen display. The use of inverse video from Integer BASIC takes on a slight aspect of "magic" in that it is supported by means of a POKE statement. The memory location numbered 50 is used in a special way in producing the APPLE II text output. Depending on what value is stored there, the characters output to the screen may be normal, inverse, or blinking. Actually, it's more complicated than that, but we won't go into it just here. All we need for now is a way to turn inverse video on and off:

```
POKE 50,63     :REM SELECT INVERSE MODE
POKE 50,255    :REM SELECT NORMAL MODE
```

Thus, the short program:

```
10 CALL -936
20 VTAB 10: TAB 15
30 POKE  50,63: PRINT "INVERSE";
40 POKE 50,255: PRINT "NORMAL"
50 END
```

will clear the screen and print the word INVERSE in inverse mode on line 10 of the screen followed by the word NORMAL in normal mode.

The Scrolling Window

The APPLE II ROM routines support the idea of a *scrolling window* for the display screen. How all the various screen support goodies interact with this concept is a little tedious, so we will try to introduce it a little at a time, coming back to it again in future sections and chapters.

The idea of a window is simple: it is a rectangular portion of the display screen which may be thought of as a subscreen within the entire screen. This is illustrated in Figure 2.1.

A window may be specified by naming four pieces of information:

Window left	the leftmost column of the window
Window width	the width in columns of the window

TABLE 2.1

Machine Language	*ROM Address*		
Routine	*Hex*	*Decimal*	*Function*
HOME	FC58	-936	Clear the text screen and home the cursor
CLREOL	FC9C	-868	Clear the screen from the current cursor position to the end of the line
CLREOP	FC42	-958	Clear the screen from the current cursor position to the end of the screen

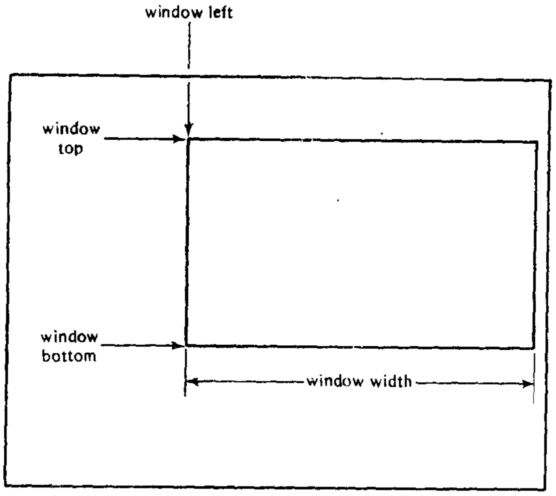


FIGURE 2.1 A Screen Window

Window top	the topmost row in the window
Window bottom	the bottommost row in the window

In APPLE's implementation of the scrolling window, these four items arr remembered in specific APPLE RAM locations within the first 256 memory locations—the so-called "Page Zero" locations. Integer BASIC allows the programmer explicitly to store values into any RAM locations desired using the POKE statement. This means that the whereabouts of the window is under the BASIC programmer's control. The magic locations are as follows:

Window left	20 (hex) or 32 (decimal)
Window width	21 (hex) or 33 (decimal)
Window top	22 (hex) or 34 (decimal)
Window bottom	23 (hex) or 35 (decimal)

Once the window parameters have been set, scrolling will be limited to the text inside the window. Put another way, any text *outside* the window will not budge when scrolling takes place.

Scrolling

When a line of text is printed at the bottom of the window, all lines of text move up one. The top line disappears out of the winJow and when the bottom line moves up, it makes room for more text to appear where it used to be.

The window parameters do not control *all* placement of text, however. VTAB statements work regardless of the window top and window bottom. TAB statements are made relative to window left and they do respect the window width.

Try the following program:

```
 1 VTAB 10: TAB 10:PRINT "*************"
 2 FOR I=1 TO 10:TAB 10:PRINT
   "*           , *":NEXT I
 3 TAB 10: PRINT "*************"
10 POKE 32,10:POKE 33,10
12 POKE 34,10:POKE 35,20
25 VTAB 1:TAB 5:PRINT "LINE 1 COL 5"
26 VTAB 10:TAB 25:PRINT "HI"
27 VTAB 23:TAB 5:PRINT "23,5"
```

This demonstrates two facts. VTAB allows you to violate the vertical extent of the window. TAB does not allow you to violate the horizontal. Why all this works the way it does is a consequence of the way the Monitor ROM routines were written. We shall not go into all the details here.

2. THE SKELETON OF A MENU-DRIVEN PRC GRAM

Before plunging into a full-length sample program, we shall discuss the idea of a *skeleton* program: an outline which will serve to guide the development of many different real programs. In order to develop the skeleton program, we need to discuss menu programs in general.

What is a Menu?

A *menu* is a list of choices, presented to the user of a program, usually on the video display. The user selects a menu item for the program to execute and the program transfers control to the appropriate section. A choice is made by pressing a key which corresponds to one of the menu selections—this could be a number or letter or even a special character—that depends on how the menu is presented.

Choosing a Personal Menu Style

Menu displays may range from the simple to the ornate. There is considerable scope here for creativity and original application of the basic screen manipulation tools.

The simplest presentation is a numbered or lettered list of short titles or phrases, each of which describes one of the options of the program. The program accepts the user's choice by reading a number or letter.

Information about what the program does or about individual choices in the menu may adorn the screen in addition to the bare text of the choices themselves.

3. CODING THE MENU-DRIVER

There are two basic facets to a menu-driver:

- Presenting the list of choices.
- Determining the user's choice and invoking the appropriate section of support in the program.

Presenting the list of choices is pretty much a matter of personal style. You will get to see one approach in the Trivia program later on. The only constraint we follow is that all the choices should fit on a single screen. If this constraint is violated, then your program is a candidate for "quick weight loss"!

Menu choices

Most menus are presented as a *numbered* or *lettered* list of choices. The program user may respond to the menu by one or two simple keystrokes. To convert this to a number which may be used to "vector" the program to the appropriate supporting code, the Integer BASIC function ASC may be used. We give the details following the presentation of the sample program.

Use skeleton programs

A *program skeleton* is a general outline from which may grow many different programs of the same nature. The skeleton may be nothing more than a standard outline which dictates what functional pieces will fit where. It may also contain a few or many *standard subroutines* which are used in many different programs. Examples of this in the APPLE Trivia program are WAIT and GET.

Figure 2.2 illustrates one possible bare-bones program skeleton for menu-driven programs. It indicates seven functional subdivisions of the program and a suggested range of line numbers to use for each.

The distribution of the line numbers is flexible and depends on the amount of program to be devoted to each subdivision. The program of Listing 2.1 is a good example of a medium-size, interactive program, written in Integer BASIC. It provides a menu with a choice of several quizzes:

(A) CHESS HISTORY
(B) OLYMPICS
(C) BOOKS AND AUTHORS
(D) COMPUTER LORE
(E) GENERAL INFORMATION

0 – 9	Banner: Title and Author
10 – 99	Declarations
100 – 999	Menu
1000 – 3999	Support Subroutines
4000 – 29999	Subject Matter Subroutines
31000 – 31999	Initializations
32000 – 32999	Introduction

FIGURE 2.2 The Skeleton of a Menu Driven Program

The section on general information has been intentionally left blank—you may fill it in with your own trivia questions.

The program's introduction indicates that there are quizzes of the following types:

1. Matching
2. Multiple Choice
3. Short Answer
4. True or False

The only type which is actually implemented by the program so far is matching. You will be given the chance to try your hand at implementing one or more of the others in the Explorations section at the end of this chapter.

Matching is relatively simple to do, since there can be no doubt about the correctness of the answers. In our case, the key to this may be found in the subroutine which scores the quiz (MARK = 1550):

```
IF ANSWERS (ORDER(I)) = I THEN RIGHT =
RIGHT + 1
```

What this means may be explained briefly as follows:

> The order in which the answers are displayed on the screen is captured in the array ORDER. Thus, ORDER(I) contains the answer to the Ith question and is printed in the ORDER(I)th position. Therefore, the ANSWER to the ORDER(I)th question is I. Confused? Think about it for a while—then if you are still confused, draw a picture or two of possible orders.

(For more info, study the following subroutines: MIXUP = 1050, SHOWANSWERS = 1100, and GETRESPONSE = 1150).

Multiple choice and true or false should also be fairly easy to implement, since the answers are exact. On the other hand, short answer quizzes must be imperfect at best—the person using the program is free to enter many different s^hort answers, all of which may in some sense be correct. The program must not only check for the "pure" answer, it must *judge* whether a given answer should be considered a *match* for the pure answer. In general, large computer-based instruction systems contain considerable amounts of intelligence aimed at the resolution of this problem.

Study the APPLE Trivia program, shown in listing 2.1, to see what you can discover for yourself. Then read the next section for a number of explanatory *programming tips* about it. Figure 2.3 shows a diagram with all the subroutines of APPLE Trivia and how they depend on or are called from one another. Such a diagram is a good reference to have when studying a program of more than one or two pages in length.

4. INTEGER BASIC CODING STYLE

There are a number of practices which can make your life considerably easier when using Integer BASIC. Consistently applying these techniques should lead to programs which are easier to read and understand. This becomes significant when someone besides the original author attempts to modify and extend the program, or when the original programmer comes back to the program after a long absence.

In this section we give a number of programming tips which explain some of the programming practices (and in some case programming tricks) which have been used in coding the APPLE Trivia program.

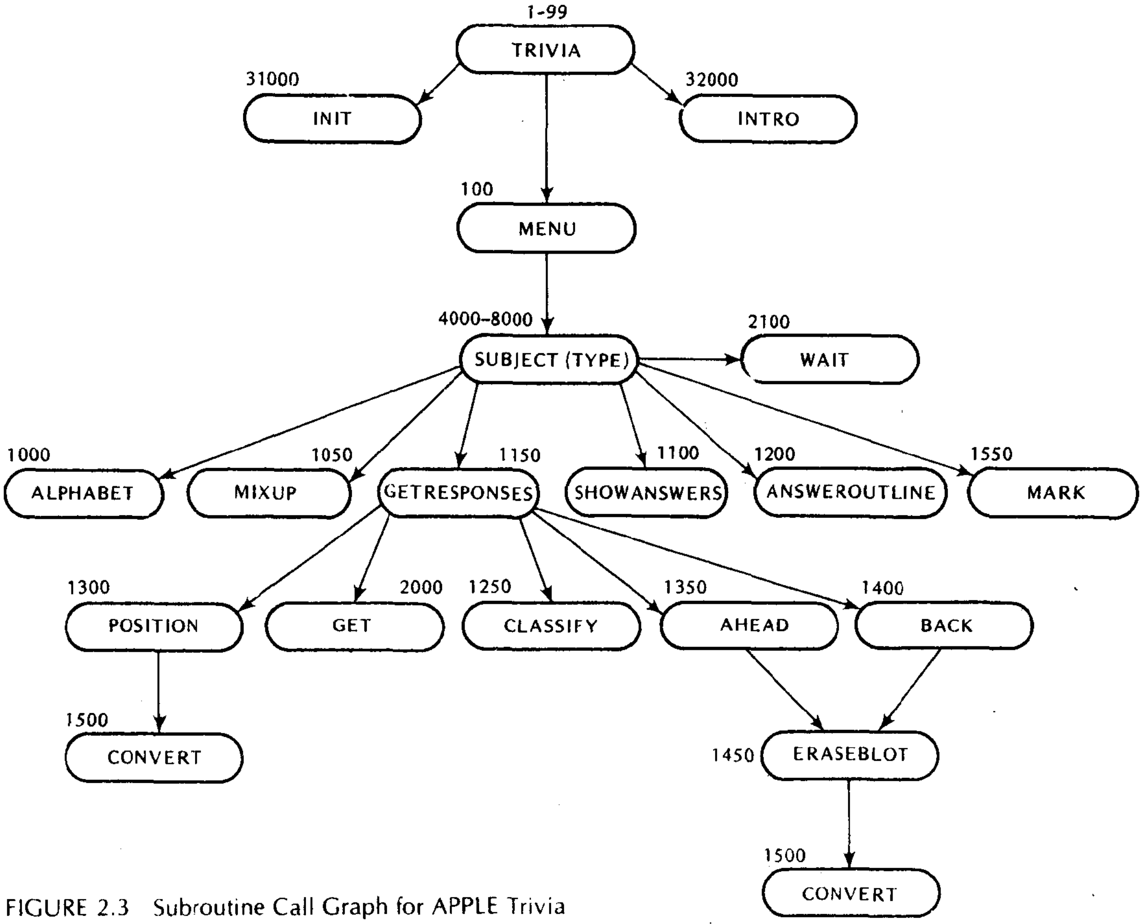


FIGURE 2.3 Subroutine Call Graph for APPLE Trivia