

# INTERMEDIATE APPLE

By Bill Parker



Take that intermediate step from elementary  
BASIC to machine language programming!

8563725

# Intermediate Apple

by  
**Bill Parker**



E8563725

Illustrated by  
**Robert J. Peters**



**DATAMOST**<sup>TM</sup>  
INC

20660 Nordhoff Street  
Chatsworth, CA 91311-6152  
(818) 709-1202



**RESTON PUBLISHING COMPANY, INC.**  
*A Prentice-Hall Company*  
Reston, Virginia

**ISBN 0-8359-3122-6**

**Copyright © 1984 by DATAMOST, Inc.**  
**All Rights Reserved**

This manual is published and copyrighted by DATAMOST, Inc. Copying, duplicating, selling or otherwise distributing this product is hereby expressly forbidden except by prior written consent of DATAMOST, Inc.

The word APPLE and the Apple logo are registered trademarks of Apple Computer Inc.

Apple Computer Inc. was not in any way involved in the writing or other preparation of this book, nor were the facts presented here reviewed for accuracy by that company. Use of the term Apple should not in any way be construed to represent any endorsement, official or otherwise, by Apple Computer Inc.

Brief excerpts from software and documentation on B.E.S.T., Edit-Soft and APLUS used by permission of Sensible Software.

Apple II is a registered trademark of Apple Computer Inc. Used by permission of Apple Computer Inc., 20525 Mariani, Cupertino, CA 95014.

Printed in U.S.A.

# ACKNOWLEDGMENTS

To Bill Sanders for the methodology and motivation to finish this book.

Dedicated to Nancy and Z. Parker, two of my closest friends.





# Table of Contents

<b>Chapter 1: Introduction to Structured Programming . . . .</b>	<b>11</b>
The Hazards of Unstructured Programming . . . . .	11
Benefits of Structured Programming . . . . .	15
The Three Control Structures . . . . .	15
The Problem with GOTOs . . . . .	19
ASA: An Alternative to GOTOs . . . . .	20
Nesting . . . . .	21
A Word on Variable Names . . . . .	22
Program List Formatting . . . . .	23
Summary . . . . .	27
For Further Reading . . . . .	29
 <b>Chapter 2: Problem Solving Using Structured</b>	
<b>    Programming . . . . .</b>	<b>31</b>
The Five Steps of Algorithm Development . . . . .	31
An Actual Example: The Shadow and the	
Building . . . . .	34
The Four Standard Program Modules . . . . .	37
Summary . . . . .	40
For Further Reading . . . . .	42
 <b>Chapter 3: Introduction To Flow Diagrams . . . . .</b>	<b>43</b>
The Basics . . . . .	43
The Control Structures . . . . .	47
Nesting . . . . .	48
Flow Diagrams vs. Flowcharts . . . . .	49
Refining Flow Diagrams . . . . .	51
Actual Example . . . . .	53
Summary . . . . .	57
For Further Reading . . . . .	57
 <b>Chapter 4: Useful Algorithms . . . . .</b>	<b>59</b>
Sort Algorithms . . . . .	60
Bubble Sort . . . . .	60
Select Sort . . . . .	61
Shell Sort . . . . .	63

DOS Algorithms .....	64
Load Directory Into An Array .....	64
RWTS .....	66
EXEC Files .....	67
Summary .....	69
For Further Reading .....	69
 <b>Chapter 5: Text Files .....</b>	<b>71</b>
Purpose of Text Files .....	71
Structure of Text Files and the Disk .....	71
Basic File Structure: Records and Fields .....	74
Sequential and Random Access Files .....	75
Text File Memory Requirements .....	77
File Design Considerations .....	79
Useful File Handling Techniques .....	81
Make a Sequential File .....	82
Read a Sequential File .....	83
Make a Random Access File .....	84
Read a Random Access File .....	85
Appending Sequential Files .....	85
Appending Random Access Files .....	86
External Sort of a Sequential File .....	88
External Sort of a Random Access File .....	90
Merge Sequential Files .....	92
Merge Random Access Files .....	93
Summary .....	95
For Further Reading .....	97
 <b>Chapter 6: Enhanced Graphics .....</b>	<b>99</b>
Limitations of Applesoft .....	99
Introduction to the HI-RES Screen .....	101
Introduction to Shapes .....	103
Shape Creating Summary .....	109
The Lunar Lander Demonstration .....	110
Creating the Lunar Lander Shape .....	110
Lunar Lander 1: Display Lander .....	112
Lunar Lander 2: Lowering the Lunar Lander .....	116
Lunar Lander 3: Display Terrain .....	117
Lunar Lander 4: Sound Effects .....	118
Lunar Lander 5: Button Control .....	119

Lunar Lander 6: Joystick/Paddle Control . . .	120
Lunar Lander 7: Explosions . . . . .	120
The Complete Lunar Lander Program . . . . .	122
Summary . . . . .	127
<b>Chapter 7: Special Printer Techniques . . . . .</b>	<b>129</b>
The Three Types of Printers . . . . .	129
Printing Out Normal Text . . . . .	130
Special Printer Commands . . . . .	131
Programming the Printer Interface Card . . . . .	133
HI-RES Screen Dump . . . . .	137
Summary . . . . .	139
For Further Reading . . . . .	140
<b>Chapter 8: PEEKs, POKEs, CALLs and Tricks of the</b>	
<b>Trade . . . . .</b>	<b>143</b>
Ampersand . . . . .	143
Applesoft Program Pointers . . . . .	143
DOS . . . . .	144
Buffers . . . . .	144
Command Error Tables . . . . .	144
Greeting Program . . . . .	146
Last Loaded File . . . . .	147
MON Flags . . . . .	147
RWTS . . . . .	147
Error Handling . . . . .	148
Game I/O . . . . .	149
Reading Paddles/Joystick . . . . .	149
Reading Pushbuttons . . . . .	150
HI-RES Graphics . . . . .	150
Shape Table Pointer . . . . .	150
Select HI-RES Page . . . . .	150
Clear HI-RES Page . . . . .	151
Display Page . . . . .	151
Page Flipping . . . . .	151
Reading the Keyboard . . . . .	152
Move Memory . . . . .	152
Reset Control . . . . .	153
Screen Control . . . . .	153
Sounds . . . . .	154
Summary . . . . .	155
For Further Reading . . . . .	155



<b>Chapter 9: How To Use An Assembler .....</b>	<b>157</b>
Advantages and Disadvantages of Assembly	
Language .....	157
A Comparison: Applesoft and Machine Code .....	157
Choosing an Assembler .....	159
Getting Started .....	161
Your First Assembly Language Program .....	162
Enhancing Your Program .....	164
Editing Your Program .....	167
Inside the 6502 .....	169
Loading Big Numbers: High and Low Bytes ....	170
Using Labels .....	171
Control Structures in Assembly Language .....	173
Basic Techniques .....	175
Sample Applications .....	178
DOS/Applesoft Problem .....	180
Summary .....	180
For Further Reading .....	182

<b>Chapter 10: Program Development Aids .....</b>	<b>183</b>
RDLN .....	183
ASA .....	187
Programming Aids by Sensible Software .....	195
Edit-Soft .....	196
APLUS .....	198
B.E.S.T. ....	201
Programming Aids by Delta Micro Systems .....	203
BASIC' .....	203
Summary .....	207
Ampersand Utilities .....	207
Applesoft Editors .....	207
Applesoft Pre-Processors .....	207
Applesoft Optimizers .....	207

<b>Chapter 11: Structures Languages .....</b>	<b>209</b>
Applesoft .....	209
Pascal .....	210
C .....	210
Summary .....	212
Bibliography .....	213

<b>Index .....</b>	<b>215</b>
--------------------	------------

# EDITOR'S INTRODUCTION

William B. Sanders, Ph.D.

This book is for the computer user who understands the fundamentals of BASIC programming and is wondering what to do next. There is a point when the novice computer user reaches a plateau, where he or she decides whether or not to learn more about computer programming. At this point, one has little alternative than to make a giant leap into the world of machine level language or into various other higher level languages. Mastering elementary BASIC is often traumatic enough to dissuade the budding programmer from risking his life on the cliffs of machine language. So the decision is often limited to taking a leap into the morass of a new language or doing nothing at all.

This book offers another choice. It is an INTERMEDIATE step that will immensely improve Applesoft BASIC programming skills and provide a whole array of proven programming techniques. Yet at the same time, it deals with the familiar constructs of Applesoft BASIC. In fact, the purpose of this book is to make programming easier, not more difficult. Rather than looking at BASIC in terms of single statements, functions or commands, it shows the user how to deal with program blocks and modules. Small, simple programs are fine for learning how to program, but there will come a point at which you will want to write a useful, fairly large program. If you've spent any time at all programming, you must have LISTed others' programs and asked yourself, "How could they keep a big program like that straight?" This book shows you how.

You may well wonder how writing larger, more sophisticated programs can be easier than writing a small simple program. For the most part, a large program is nothing more than a well organized set of small programs, and the key to that is organization and structure. First, rather than rewriting an entire program every time you sit down at the computer this book shows you how to save and then re-use program modules that can be employed in several programs. With only a few program lines, it is possible to connect several previously written modules into a larger program. Thus, the larger program is actually simpler than blindly piecing together a small one. Also by using structured programming techniques, you will be able to see more clearly what you have done. For anyone who has written a huge

program and then gone back to it, one is often at a loss to remember what everything does and why it does it. By clearly documenting and arranging it, any Applesoft program can be made clearer. Mr. Parker even provides some utilities to assist in making Applesoft more lucid.

In addition to showing the user how to attack a programming problem, this book shows how to master some of the more advanced features of Applesoft. In my introductory book, *The Elementary Apple*, I only wanted to get a new user started in programming. The book was for someone who brought home their computer and wanted to get started without too many tears. However, DATAMOST wanted a book that would take the next step in programming the Apple Computer. Therefore, many of the features that were just touched upon in *The Elementary Apple*, such as graphic animation, shape tables, POKEs, PEEKs and disk file handling, are explored in depth here.

Next, the book takes the first step toward advanced programming. It is not the giant leap described above, but rather it is an introduction to assembly language programming. The best way to introduce an assembly level program is to explain how to use an assembler, and that is exactly what it does. This gives the user a chance to take a look before the leap. You will be shown how to get an assembler up and working, along with some sample assembly level programs. Everything will be kept simple, and while you cannot expect to become an expert at assembly level programming, you will learn enough to get started.

Finally, the book concludes with a number of utility programs provided by the author, along with some suggested commercial programs. All of these programs are utilities to make programming easier, clearer and more efficient.

## MEET THE AUTHOR

Bill Parker has the ideal background for creating a book such as this. With a solid background in both journalism and computers, he has both the ability to communicate clearly in English and to write computer programs. Bill is a staff writer for *Call -A.P.P.L.E.*, and former editor of the *Sandy Apple Press*, the club magazine of Apple Corps of San Diego. He has taught computer courses in both the University of California, San Diego and at San Diego State University Extension programs. He is currently a full-time computer consultant and writer.

# CHAPTER 1

## INTRODUCTION TO STRUCTURED PROGRAMMING ON THE APPLE

This book will take you from the point of being a fledgling Apple programmer and show you some important principles that can help you handle more complicated programming problems.

The method emphasized here is a technique known as *structured programming*: a “one-step-at-a-time” method of reducing big problems into smaller, more manageable ones.

### The Hazards of Unstructured Programming on the Apple

Consider the following two programs:

**Program 1:**

```
100 I = I + 1
110 GOSUB 200
120 GOTO 100
130 :
200 PRINT I
210 IF I < 100 THEN 120
220 RETURN
```

RUN

1  
2  
3  
4  
5  
6  
7

8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25

?OUT OF MEMORY ERROR IN 210

**Program 2:**

```
100 FOR I = 1 TO 200
110 GOSUB 200
120 NEXT I
130 :
200 PRINT I
210 IF I < 100 THEN 120
220 RETURN
```

RUN  
1

?NEXT WITHOUT FOR ERROR IN 120

Why are these programs generating nonsense error messages? After all, little Program 1 couldn't possibly consume all of the Apple's memory and line 100 of Program 2 contains a FOR statement as plain as the nose on your face. So why the errors?

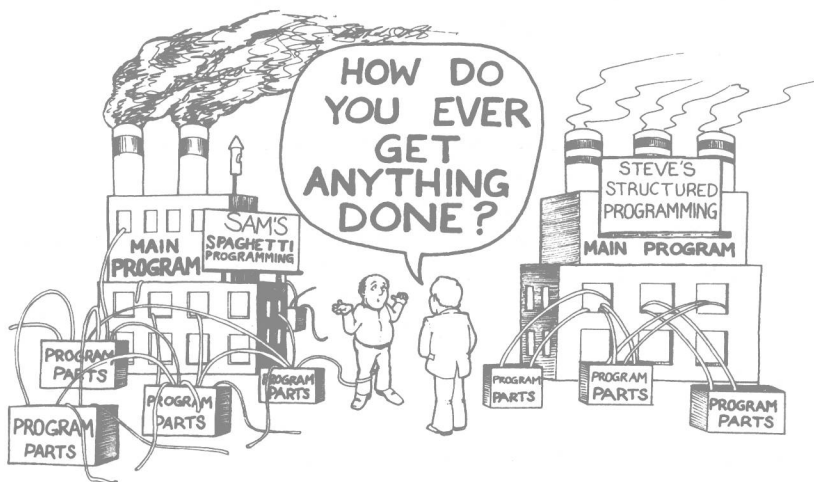
These programs are typical examples of *unstructured programming*, a kind of a programming “by the seat of your pants” approach to problem solving. The problem with it is that it just doesn’t follow the way your computer “thinks,” and the error messages show it.

Another problem with this type of programming is that it is unclear. The bigger the program becomes, the harder it is to read and understand. This becomes an even bigger problem when someone unfamiliar with your program needs to enhance, customize or just learn from it. In extreme cases, *you may not even be able to understand your own program*, which could occur if you come back to it after a few months. Your memory of the intricate portions of your program will be gone, and you will then be unable to make sense of the more tangled portions of your own code.

Some computer scientists in the 1960’s noticed these things and decided to take a long, hard look at the state-of-the-art in programming at the time. This is what they found:

- There was no organized, systematic way of even approaching a programming problem. Good programmers simply “dove in” (sound familiar?) and came up with something that worked — most of the time. Bad programmers floundered about even longer.
- There was no real assurance that a program was written “correctly” and would be reliable.
- There was no standardized vocabulary for describing the way a problem was turned into a program (this is known as *algorithm* development). You had to be intimately familiar with the computer language chosen by the program author to understand how he solved the problem.
- Programming projects became stalled for expensive periods of time when old programmers left the project and new ones came in to replace them. It took a certain amount of time just to be able to understand what the previous programmer had done and then to be able to continue from that point.

- Flowcharts were worthless. Although they were supposed to be done at the beginning of a problem to provide the programmer with an easy-to-follow graphical representation of the steps necessary to write the program, they were frequently left (if done at all) until after the program was up and running. This happened because the more complicated a program became, the more complicated (and unclear) the flowchart became.
- Trying to read a long program in an effort to understand how it worked was an exercise in futility, because control branched all over (through the use of the ubiquitous GOTO statement). This method of programming, which also prevented large programs from being broken into smaller, more manageable “modules,” came to be known as *spaghetti programming*.



This sorry state of affairs led Edsger Dijkstra, one of the “Fathers of Structured Programming,” to remark in the preface to his book, *A Discipline of Programming*:

... on the one hand I knew that programs could have a compelling and deep logical beauty, on the other hand I was forced to admit that most programs are presented in a way fit for mechanical execution but, even of any beauty at all, totally unfit for mechanical appreciation.

Fortunately, there is help! The same group of scientists who analyzed the problems caused by “normal” programming practices also came up with a way to avoid them. This method allowed programmers to write programs with “compelling and deep logical beauty” and has come to be known as structured programming.

## Benefits of Structured Programming

We will now investigate the basic elements of structured programming in Applesoft. Among its benefits are:

1. Programs are more easily understood. Programs are easier to read and the logic flow is easier to follow.
2. The possibility of making errors is reduced. This is known as “anti-bugging.” You may also have heard of this concept in the well-known commercial where it is stated, “problems are built out — not in.”
3. Programs are easier to maintain. This makes it easier for anyone to understand, improve or enhance your program.
4. It's faster to code with structured programming. The logic is simple and straightforward.
5. It provides an easier transition to other higher level languages. Currently, most programs written with the aid of higher level languages use structured programming techniques.
6. It's easier to code large programs. Large programs can be broken into “modules” and given to different programmers for development. This speeds up the development of the program and makes it easier to coordinate the finished modules into one system.

## The Three Control Structures

The benefits of structured programming are the direct result of a key discovery by researchers:



No matter how complicated a program is and no matter in which language a program is written, any program can be written with just three basic *control structures*: sequence, decision and loop.

As the name implies, control structures tell the computer which instruction to execute next. It's how you control the logic flow in your program that determines how clear and manageable the program is. Let's take a look at some examples of each kind of control structure:

## SEQUENCE

### True Control Structures

### Applesoft

```
100 X = 3
```

```
110 Y = 6
```

```
120 PRINT "THIS IS A TEST"
```

```
100 X = 3
```

```
110 Y = 6
```

```
120 PRINT "THIS IS A TEST"
```

Here, you can see that there is no difference between true control structures and the structures that Applesoft provides. This is because sequential statements are the most basic commands that can be used on a computer; all languages must use them.

## DECISION

### True Control Structures

### Applesoft

```
100 IF X > Y
```

```
    THEN PRINT "X > Y"
```

```
100 IF X > Y
```

```
    THEN PRINT "X > Y"
```

This kind of IF THEN decision is the most basic decision structure available; all languages have this structure in one form or another.