$14.95

eston
blishing
mpany
Inc.

INTERMEDIATE APPLE

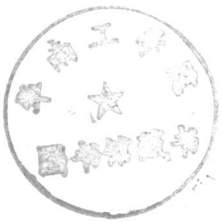
By Bill Parker

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

Intermediate Apple

by
Bill Parker

Illustrated by
Robert J. Peters



DATAMOST™
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

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

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 progam 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.

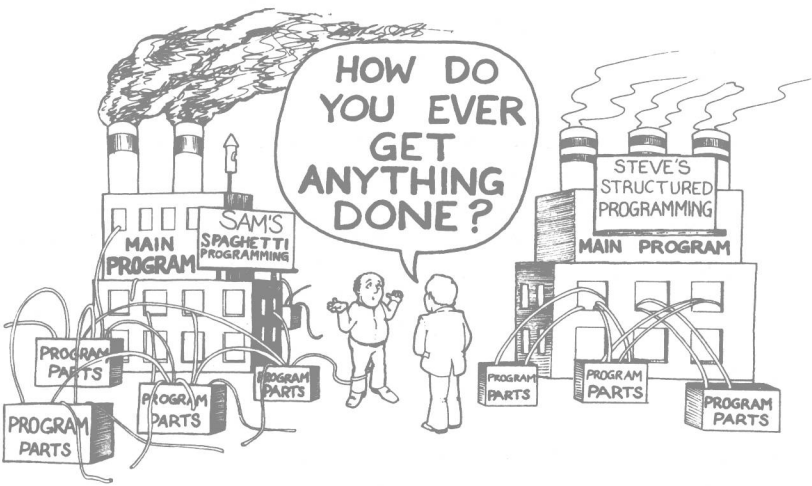
- There was no real assurance that a program was written "correctly" and would be reliable.

- 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

```
100 X = 3
110 Y = 6
120 PRINT "THIS IS A TEST"
```

Applesoft

```
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

```
100 IF X > Y
    THEN PRINT "X > Y"
```

Applesoft

```
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.