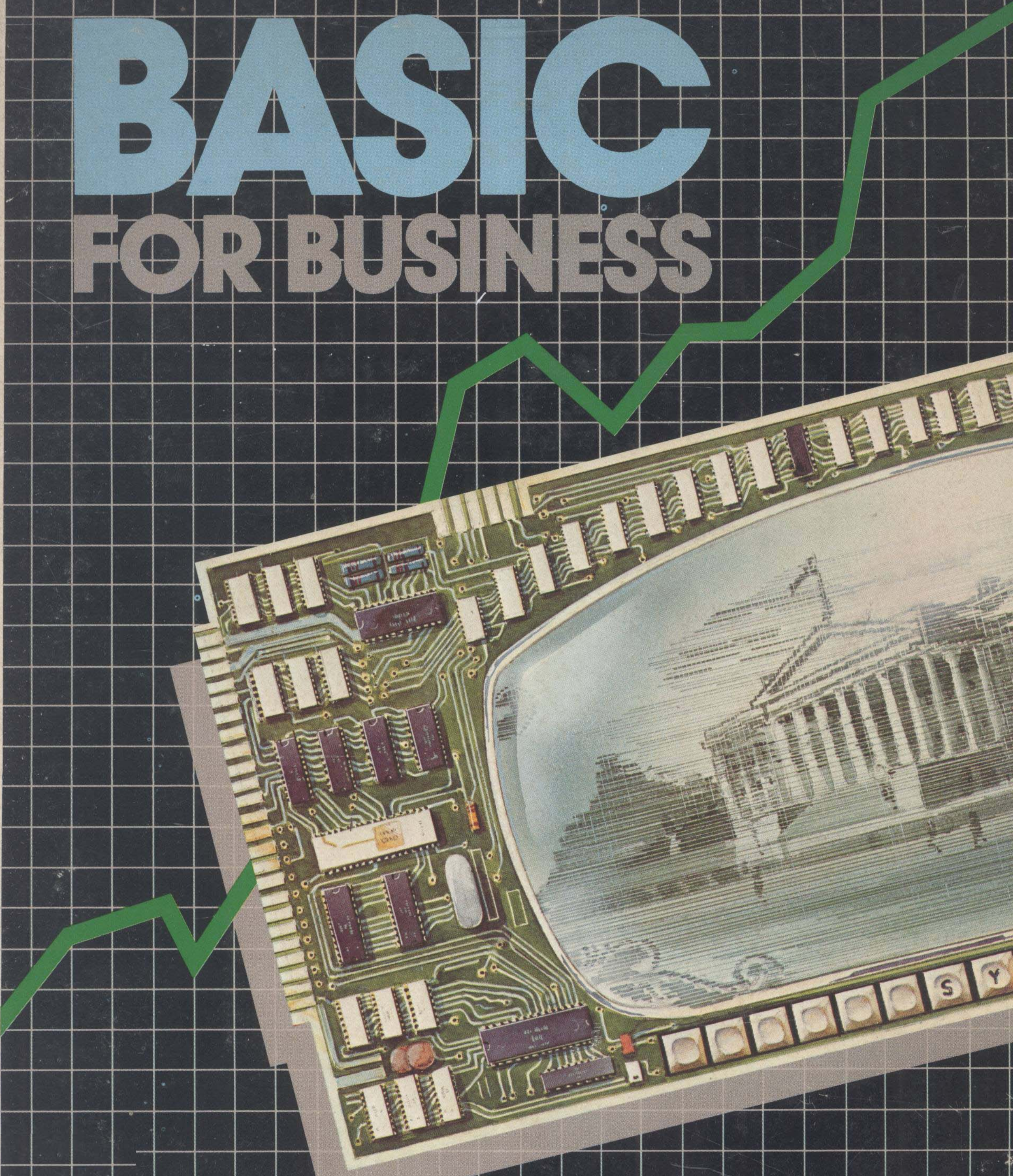
BASIC FOR BUSINESS

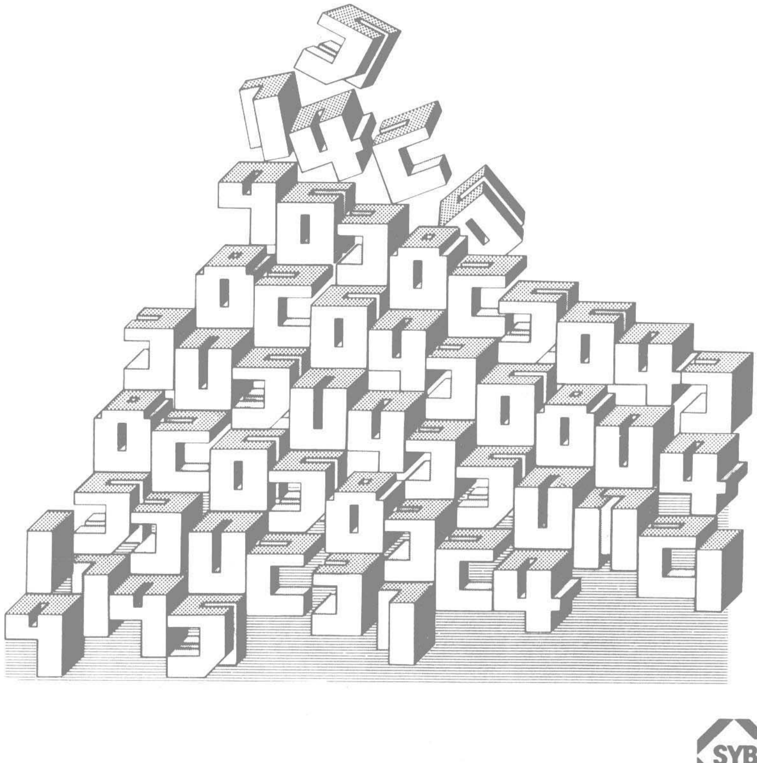


Douglas Hergert

SYBE

BASIC for Business

Douglas Hergert



SYBEX

Berkeley • Paris • Düsseldorf

CREDITS
Cover illustration by Daniel Le Noury
Layout and technical illustrations by Jeanne E. Tennant

Library of Congress Card Number: 81-85955
ISBN 0-89588-080-6
First Edition 1982
Printed in the United States of America
10 9 8 7 6 5 4 3 2

Acknowledgements

Several people contributed significantly to this book by reading the manuscript and making thoughtful suggestions. These include James Compton, whose editorial sense of style and clarity exerted a positive influence throughout this book; Mark Evanoff, Richard Hergert, Rudolph Langer, and X.T. Bui, who carefully reviewed the technical content of the manuscript; and Elaine Foster and Valerie Brewster, whose questions and comments helped me see the manuscript from the reader's point of view.

Thanks are also due to Bret Rohmer and his fine production staff. Jeanne Tennant produced the layout and the technical drawings; Judy Wohlfrom was responsible for the pasteup; Mati Sikk performed his usual miracles at the phototypesetting equipment; and Hilda van Genderen and Cheryl Wilcox painstakingly proofread the book through each stage of production. The imaginative cover art is by Daniel Le Noury.

Introduction

COMPUTER LITERACY is fast becoming an essential qualification for anyone working in the business world. As more and more business tasks are aided or directed by computers—from word processing to decision making—business professionals at every level are learning more about the powers and limitations of this ''new machine'' in our midst.

One way to find out what a computer can do is to learn to program. This book is a business-oriented introduction to computer programming in BASIC, which continues to be the most popular language available for microcomputers. In this simple, yet surprisingly powerful language, beginning programmers can quickly learn to write significant application programs.

The goal of this book, then, is to train business professionals to read, write, and ''debug'' BASIC programs for business applications; that is, to help them become literate in business computing. Each chapter focuses on one syntactical feature of BASIC and presents at least one significant program to illustrate how that feature is used.

The program examples are all presented for their instructional value. This book is not a collection of ready-made, ''black box'' programs offered without explanation, but rather a guide to writing usable programs. Many programs are accompanied by brief descriptions of the business tasks they are designed to perform. These familiar accounting tasks are summarized to help explain what the programs do, and to demonstrate the importance of *defining* a problem before writing a program to solve it.

Chapter 1—*A First Look at BASIC*—provides an introductory overview of the programming concepts and vocabulary covered in this book. A first program is presented (the cost-of-goods-sold program), though emphasis at this point is on understanding the general organization rather than the specific details of the program.

Chapter 2—*Beginning Concepts*—introduces input and output in BASIC. Special features such as **TAB** and **PRINT USING** are discussed in detail; these features aid the programmer in producing attractive and readable reports. (We also discuss how to get along without these features in a version of BASIC that does not have them.) Additional topics of this chapter are variables, assignment statements, operations, decisions, and transfer of control. All this new material is brought together in the comparative income statement program, presented and discussed in detail in this chapter.

Chapter 3—***FOR*** *Loops*—describes iteration in BASIC; that is, how to instruct the computer to perform the same command many times. A monthly sales report program is presented in two forms, first to produce a simple report, and then to produce a bar graph of monthly sales.

Chapter 4—*Arrays*—explains how to organize groups of related data conveniently in BASIC. An array is a *data structure* that requires precise syntax and a certain conceptual background, which is provided in this chapter. The power of arrays is illustrated in the present value program.

Chapter 5—*Subroutines and Program Structure*—moves step by step through a simplified general ledger program to teach the important concepts of *top-down, modular* programming. This chapter shows how to write programs that are readable and easily debugged.

Chapter 6—*Arithmetic Functions*—begins with a description of built-in functions and how they might be used in business applications. Following this is an introduction to user-defined functions, which allow the programmer to create arithmetic functions that are not included in the language itself. Among the several programs in this chapter is one that produces a break-even point graph for cost-volume-profit analysis.

Chapter 7—*Strings*—presents a number of functions that operate on, or return information about, strings of characters in BASIC. Two programs are developed for exploring the ASCII code. *Sorting* (i.e., arranging information in a given order) is also introduced, and the personnel list program illustrates the concepts presented in this chapter.

The programs in this book were written and developed on TRS- 80 and Apple II computers. Many minor differences between "versions" of BASIC are noted, and you are often directed to see how your BASIC handles a particular feature. The best place to read this book is in front of your own computer, so you can try out each program as it is presented in the book.

Exercises at the end of each chapter will guide you in further exploring the characteristics of BASIC and in approaching new applications. Appendix A contains suggested answers to some of these exercises, including several new programs.

An additional feature of this book will help you place your new understanding of BASIC in the broader context of business programming. Short optional sections appear near the end of every chapter, comparing the features of BASIC with other languages that are familiar to the business world—COBOL, Pascal, and FORTRAN. The goal is to clarify some of the relative advantages of each of these languages and to point out what they have in common with each other and with BASIC. To complement these chapter-by-chapter descriptions, Appendix B presents one complete program in each of these languages. While none of this is meant to replace formal introductions to COBOL, Pascal, and FORTRAN, this material may help you decide what language to study after mastering BASIC.

The *reserved words* of all four of the languages discussed in this book have been set in **boldface** type.

All in all, this book is designed to demystify programming for the business professional, and to serve as a guide to beginning business programming in BASIC.

Table of Contents

CHAPTER 1

A First Look at BASIC

APPROACHING A FIRST COMPUTER LANGUAGE, we are faced with learning *two* new sets of vocabulary. One set involves the programming language itself, which has not only its own vocabulary, but also its own syntax and structure. The second set of vocabulary comprises all the terms that computer programmers use when they talk about the details of their work. In Chapter 1 we will begin learning some of these terms, even as we take a first broad look at BASIC. Our aim will be to prepare for a more detailed investigation of BASIC programming in later chapters of this book.

First BASIC Program: Cost of Goods Sold

The program we will be studying in this chapter performs three main tasks:

- It *prompts* the user to type in the inventory and sales data that the program requires as input.
- It calculates two new values—the cost of goods sold (which we will refer to as COGS), and the gross margin on sales—from the input data.
- Finally, it displays a well-organized report of both the input data and the calculated values.

Although this program is simple, it will serve to illustrate a number of important concepts; examining it, we will begin to understand how programs are written and organized in BASIC. An accompanying *flowchart* will help us conceptualize the *control structures* of the language. An examination of the output from this program will lead us to our first discussion of one of the continuing themes of this book: the importance of planning clear, well-organized and attractive output formats when programming for business applications.

The program *listing* appears in Figure 1.1. (A listing is simply a display of the lines of a program.) The first thing to notice is that all the lines of a BASIC program are numbered. The *lines of code,* as we sometimes call them, are always numbered in ascending order, although the numbering need not be in uniform multiples. In the COGS program the numbers are all multiples of 10, but this is an arbitrary choice. They could just as well have been in multiples of 5 or 50 or 100, or in uneven (ascending) intervals. It is good programming practice, however, to leave ''room'' between lines for insertion of new lines of code. This is because we often find ourselves adding lines to programs for one purpose or another. If the lines of the COGS program had been numbered sequentially from 1 to 65, it would be impossible to insert lines anywhere inside the program.

Another fact that we notice right away about this program is that most of the lines begin with the words **REM**, **INPUT**, or **PRINT**. All three of these words are part of the BASIC vocabulary, or what might be called the *reserved words* of the language. The prevalence of

INPUT and **PRINT** lines indicates what we have already mentioned about this program—that it mainly performs input/output operations. The **REM** lines are *rem*arks, or comments, about the program; this feature of BASIC merits some discussion.

```
10   REM    COGS PROGRAM
20   REM    D. HERGERT             8 AUGUST 1981
30   REM
40   REM    VARIABLE NAMES
50   REM    I                      INCOME ON SALES
60   REM    B                      BEGINNING INVENTORY
70   REM    P                      PURCHASES DURING PERIOD
80   REM    E                      ENDING INVENTORY
90   REM    C                      COST OF GOODS SOLD
100  REM    G                      GROSS MARGIN ON SALES
110  REM    A$                     ANSWER STRING
120  REM
130  REM    INPUT SECTION
140  REM
150  INPUT "INCOME ON SALES"; I
160  INPUT "BEGINNING INVENTORY"; B
170  INPUT "PURCHASES DURING PERIOD"; P
180  INPUT "ENDING INVENTORY"; E
190  REM
200  REM    CALCULATION OF COGS AND GROSS MARGIN
210  REM
220  C = B + P - E
230  G = I - C
240  REM
250  REM    OUTPUT SECTION
260  REM
270  F$ = "$$##,#####.##"
280  PRINT : PRINT
290  PRINT TAB(30); "COGS" : PRINT : PRINT
```

Figure 1.1: The Cost of Goods Sold (COGS) Program

```
300  PRINT "INCOME ON SALES"; TAB(42);
310  PRINT USING F$; I
320  GOSUB 600
330  PRINT "BEGINNING INVENTORY"; TAB(30);
340  PRINT USING F$; B
350  PRINT "PURCHASES DURING PERIOD"; TAB(30);
360  PRINT USING F$; P
370  GOSUB 600
380  PRINT "GOODS AVAILABLE FOR SALE"; TAB(30);
390  PRINT USING F$; B + P
400  PRINT "ENDING INVENTORY"; TAB(30);
410  PRINT USING F$; E
420  GOSUB 600
430  PRINT "COST OF GOODS SOLD"; TAB(42);
440  PRINT USING F$; C
450  PRINT "GROSS MARGIN ON SALES"; TAB(42);
460  PRINT USING F$; G
470  GOSUB 630
480  INPUT "ANOTHER SET OF DATA"; A$
490  IF (A$="Y") OR (A$="YES") GOTO 150
500  STOP
600  PRINT TAB(30); "---------------------------"
610  PRINT
620  RETURN
630  PRINT TAB(30); "==========================="
640  PRINT
650  RETURN
```

Figure 1.1: The Cost of Goods Sold (COGS) Program (cont.)

REM: Documenting a BASIC Program

The lines that begin with **REM** are not part of the programmer's instructions to the computer. Rather, they are an aid to anyone who might someday want to understand the program (including the original programmer six or eight months after the program is written). Anything at all can be written on the **REM** lines, and there is no "standard" way of writing comments for a program. Although BASIC programs are not especially hard to read, some features of the language can cause confusion and complicate the task of figuring out what the program does. A good BASIC programmer will recognize these difficult features and *document* them—in the form of **REM** comments—for the benefit of anyone who might, in the future, need to understand how the program is organized.

In the COGS program we see several groups of **REM** lines. The first lines identify the program, the author, and the date the program was written. The date can be an important piece of information if the program is ever revised and distributed in several different versions. Anyone who has a copy of the program should be able to tell what version it is.

The next group of **REM** lines identifies the *variables* that are used in this program. We will have more to say about variables later in this chapter; for now we only need to know that variables are used to store values used in the program. Every variable has a name; however, in most versions of BASIC, variable names are restricted to two or three characters. Other computer languages allow us to write *meaningful* variable names such as these:

```
NET-INC
NET_INCOME
NINCOME
```

In BASIC, we are more likely to be restricted to a single letter:

```
N
```

or a single letter followed by a digit:

```
N1
```

Since this is the case, it is often extremely valuable to provide a list of the variables used in a program, and brief explanations of what they are used for. This is the purpose of the **REM** lines under the heading VARIABLE NAMES.

Finally, we see three **REM** comments that identify sections of the program—INPUT SECTION, CALCULATION OF COGS AND GROSS

MARGIN, and OUTPUT SECTION. The words *section, routine,* and *block,* which often have specific meanings in other programming languages, are used somewhat informally in descriptions of BASIC to refer to a group of lines designed to perform a particular task. The **REM** comments of the COGS program thus divide the program into three different sections, making its organization immediately evident.

In summary, **REM** lines may be used (or not used) according to the immediate needs of the programmer and the anticipated needs of those who will later be reading the program. Each programmer must judge how much documentation is appropriate and necessary. We should note that **REM** comments do take up some space in the memory of the computer. This may be a disadvantage in a particularly long program if memory space is limited. Nevertheless, **REM** comments are strongly recommended as an important part of any BASIC program.

BASIC Input/Output

BASIC is an *interactive* programming language, which means that we can type information into the computer from the keyboard *while the program is being executed.* Another way of saying this is that a program *run* involves a *dialogue* between the *user* and the computer, guided by the program instructions. A programming language that is not interactive, that requires all data to be input before the beginning of the program run, is said to run in *batch* mode. FORTRAN and COBOL, two languages that we will periodically examine in this book, were originally designed as batch-mode languages.

The word in a BASIC program that creates dialogue is **INPUT**. We will be studying the details of this instruction in Chapter 2. For now, notice that the COGS program has four **INPUT** lines: lines 150 to 180. As we will see, each of these lines produces a prompt that tells the user what data to type into the computer. For example:

```
BEGINNING INVENTORY?
```

The phrase "BEGINNING INVENTORY" lets the user know exactly what data to input at this point in the dialogue. The question mark is supplied by many versions of BASIC so that the user knows that the program is waiting for input.

The **PRINT** statement, another instruction we will be mastering in Chapter 2, produces *output* from BASIC. Depending on what kind of output device the computer is connected to, the **PRINT** statement might produce a line of text on a screen or an actual line of print on a printer.

The **PRINT** statement can appear as the only word of a line of code. It then produces a blank line in the output. Notice line 280 of the COGS program:

```
280   PRINT : PRINT
```

The colon in BASIC separates two instructions in a single line of code. (Some versions of BASIC use the backslash (\) instead of the colon.) Line 280 thus produces two blank lines in the output.

When we run our COGS program to examine a sample dialogue and the resulting output, we will see the actual results of the **PRINT** and **INPUT** statements. Before we do, however, we must master a few more programming terms.

Versions of BASIC

We have referred several times to different *versions* of BASIC. Since we are about to run our first program, this is a good opportunity to discuss the meaning of that expression. The explanation will divert us from our program for a moment.

BASIC, like FORTRAN, COBOL, and Pascal, is a *high-level* language. The instructions of these languages are in English, making programming relatively "natural" for human beings. However, the *machine code,* representing the instructions that the computer actually carries out, is not in English or any other human language, but rather in strings of 0s and 1s called *binary code.* The link between a high-level language and machine language is a program itself; this program is called either a *compiler* or an *interpreter,* depending on how it ultimately supplies the machine code instructions that the computer will perform.

A compiler translates the entire program in one pass and, as long as there are no *bugs* (mistakes) in the program, creates a set of binary instructions that can be executed directly by the computer. An interpreter, on the other hand, works on one line of the program at a time,