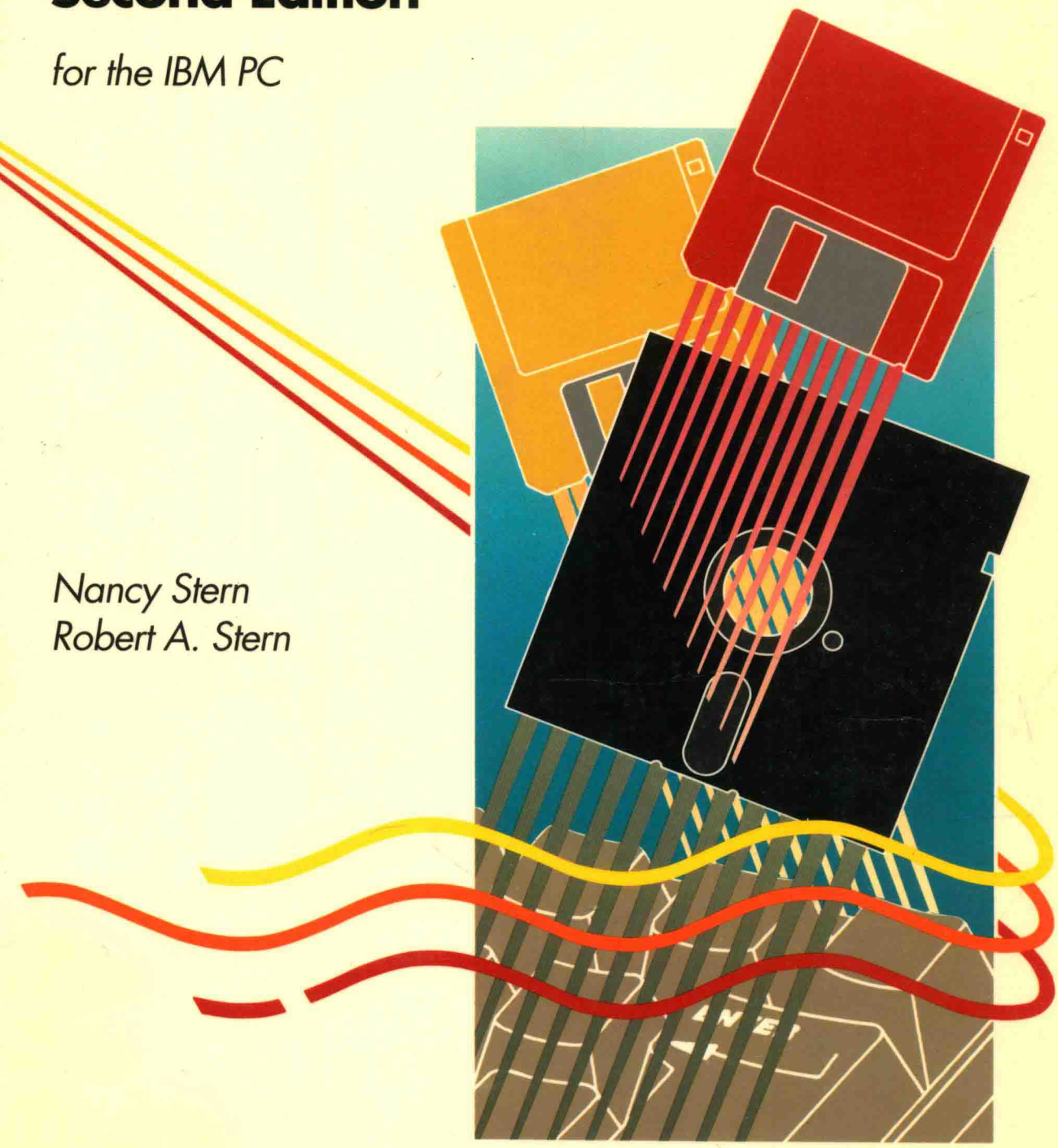
Getting Started with Structured BASIC

Second Edition

for the IBM PC



Nancy Stern
Robert A. Stern

A Wiley PC Companion

Getting Started with Structured BASIC

Second Edition

Nancy Stern
Hofstra University

Robert A. Stern
Nassau Community College

John Wiley & Sons, Inc.
New York Chichester Brisbane Toronto Singapore

ACQUISITIONS EDITOR Beth Lang Golub
MARKETING MANAGER Carolyn Henderson
PRODUCTION SUPERVISOR Charlotte Hyland
MANUFACTURING MANAGER Andrea Price
COPY EDITING SUPERVISOR Deborah Herbert

This book was set in Garamond Light by GTS Graphics and printed and bound by Hamilton Printing. The cover was printed by Lehigh Press.

Recognizing the importance of preserving what has been written, it is a policy of John Wiley & Sons, Inc. to have books of enduring value published in the United States printed on acid-free paper, and we exert our best efforts to that end.

ISBN 0-471-58709-5

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Getting Started with Structured BASIC

Wiley PC Companions

Stern/Stern: GETTING STARTED WITH STRUCTURED BASIC, Second Edition
Murphy: GETTING STARTED WITH DOS 5.0
Russakoff: GETTING STARTED WITH WINDOWS 3.1
Kronstadt/Sachs: DISCOVERING MICROSOFT WORKS 2.0
Murphy: GETTING STARTED WITH WORDPERFECT 4.2/5.0**
Murphy: GETTING STARTED WITH WORDPERFECT 5.1
Murphy/Potter: GETTING STARTED WITH LOTUS 1-2-3, RELEASE 2.2
Farrell: GETTING STARTED WITH LOTUS 1-2-3, RELEASE 2.3
Murphy: GETTING STARTED WITH QUATTRO**
Arnolds/Hammonds/Isham: GETTING STARTED WITH dBASE III PLUS**
Gaylord: GETTING STARTED WITH dBASE IV
Wiley: EXPLORING DOS, WORDPERFECT 5.1, LOTUS 1-2-3 (RELEASE 2.2), AND dBASE III PLUS

** Educational version software available

Wiley Macintosh Companions

Abernethy, Nanney and Porter: EXPLORING MACINTOSH: Concepts in Visually Oriented Computing
Nanney, Porter and Abernethy: EXPLORING MICROSOFT WORKS 2.0—Macintosh

To Melanie and Lori

PREFACE

This book, which is part of the Wiley *Getting Started* series on software tutorials, focuses on structured BASIC programming for IBM PCs and their compatibles. The book can be used as a self-teaching guide or as a tutorial in conjunction with (1) an introductory course in computing, or (2) a BASIC programming course. No prior programming or computing experience is needed.

This book covers the fundamentals of BASIC programming, not the advanced features of the language. The syntax explained in the text is compatible with virtually every PC version of BASIC; where there are fundamental differences, they are specifically addressed in the text. In addition, the programming concepts described in the text are common to all versions of BASIC and, indeed, to all programming languages. We use pseudocode throughout the book as a program planning tool. We have written a relatively short introduction to BASIC that will enable students to write elementary- and intermediate-level programs in a relatively short time.

This book includes many of the pedagogic tools common to our other computing texts. Each chapter is in outline form and has a step-by-step presentation of material followed by numerous examples. Most chapters have self-tests with solutions at key points within the chapter, full chapter self-tests, and review questions that can be assigned for homework.

Many BASIC books claim to present a structured version of the language but actually begin with nonstructured examples or with a focus on syntax that fails to address structured programming concepts. From the very onset, we focus on BASIC programs that handle a variable amount of input and that are fully structured.

In addition to emphasizing structured concepts, we focus on other programming techniques for making programs easier to read, debug, maintain, and modify. These include top-down and modular programming concepts and stylistic features such as useful naming conventions and conventions for displaying meaningful comments.

We thank the following people at John Wiley for their assistance and support in the preparation of this book:

Editorial—Beth Lang Golub, Bill Oldsey, Bonnie Lieberman
Production—Charlotte Hyland, Lucille Buonocore, Ann Berlin
Marketing—Carolyn Henderson, Steve Kraham
Proofreading—Shelley Flannery, Suzanne Ingrao
Copy Editing—Mary Konstant

We thank the following people for their assistance in reviewing the manuscript:

Gary Baker
Marshalltown Community College

Virginia Gregory
Hofstra University

Marilyn Meyer
Fresno City College

Bill Smith
Nassau Community College

Floyd Winters
Manatee Community College

Finally, we express our appreciation to our assistant, Carol L. Eisen, for her assistance in preparing the manuscript.

If you have any questions, comments, or suggestions regarding this book, please contact us through our editor, Beth Lang Golub, at John Wiley and Sons, Inc., 605 Third Avenue, New York, NY 10158, 212-850-8619. We can also be reached via Bitnet at ACSNNS@HOFSTRA, or via Internet at ACSNNS@VAXC.HOFSTRA.EDU.

Nancy Stern
Robert A. Stern

CONTENTS

1

The Nature of BASIC

BASIC as a Universal Language

BASIC is designed to be an easy-to-learn programming language that you can use for writing programs in many applications areas. The term *BASIC* is an acronym for *B*eginner's *A*ll-Purpose *S*ymbolic *I*nstruction *C*ode. When it was developed in the early 1960s by John Kemeny and Thomas Kurtz at Dartmouth College, BASIC was one of only a few existing symbolic programming languages. It gained immediate and widespread popularity because it is easy to learn, easy to use and, at the same time, powerful and flexible.

Today, there are dozens if not hundreds of versions of BASIC available for microcomputers as well as mainframes. In fact, when micros were introduced in the 1970s, BASIC was the first and, in most instances, the only symbolic programming language available for them. It remains the most widely used symbolic language for microcomputers, and it is still used on mainframes as well.

Most Commonly Used Versions for Micros

Although there are numerous versions of BASIC for micros, several are predominant in the current marketplace. Note that PC-DOS is the version of DOS for IBM personal computers (PCs), and MS-DOS is a comparable version for IBM-compatible PCs.

Most Common Versions of BASIC for IBM PCs and IBM-Compatible PCs

Name	How to Acquire	Features
QBASIC	Comes with PC-DOS 5 and higher	Can be used on IBM PCs and IBM-compatible PCs
BASICA	Comes with PC-DOS 4 and lower	Can be used only on IBM computers and *some* compatibles
QuickBASIC	Developed and sold by Microsoft for under $100	Can be used on IBM PCs and IBM-compatible PCs

GW-BASIC	Used with MS-DOS, most often bundled with it; can be purchased for under $100	An older version of BASIC for IBM-compatible PCs; similar to BASICA for IBM computers
BASIC	Comes with PC-DOS 4 and lower	Very limited version just for IBM PCs

Most inexpensive versions of BASIC—like those just mentioned—are interpreted, not compiled. Only QuickBASIC, the most advanced of the BASIC versions discussed in this text, gives you an option of either compiling or interpreting your program.

Compiled versions of a language save the translated, machine language code in a program file that can then be executed over and over again. Because the need for repeated translations is eliminated, compiled versions of BASIC are more useful for programs that are to be run repeatedly than are interpreted versions, which must be retranslated before each program run. For example, programs to be run on a regular production basis are typically compiled. In addition, compiled versions of BASIC tend to translate and run faster. Most compiled versions of BASIC, like QuickBASIC, are fully compatible with QBASIC and BASICA. This means that programs written in QBASIC or BASICA can be translated into QuickBASIC and then executed with little or no modification by means of QuickBASIC compilers.

QBASIC (for PC-DOS and MS-DOS 5.0 and higher) and BASICA (for PC-DOS 4 and lower on IBM computers) are typically bundled with the DOS disks themselves so that their actual cost is negligible. As a result, they have been the most widely used versions for IBM micros and their compatibles. QBASIC will likely become as popular in the years ahead because it is bundled with all newer versions of DOS.

Compiled versions of BASIC, including QuickBASIC, must be purchased separately, but they typically sell for under $100.

QBASIC is significantly more advanced than BASICA and has a menu format that is more user-friendly. In this text we consider both QBASIC and BASICA in detail and highlight their differences. We discuss the even more advanced QuickBASIC as well. As noted, BASICA programs can easily be converted to QBASIC or QuickBASIC format later on.

We focus on a common core of QBASIC and BASICA instructions that are likely to run, with perhaps minor adjustments, using any version of BASIC.

To load in a version of BASIC, type the program name (QBASIC, BASICA, QB, and so on) and press the Enter key. You are then ready to enter a program.

With QBASIC, a full-screen display appears after you type QBASIC. See Figure 1.1. Press the Esc key to clear the welcome message so that you can begin entering a program.

With BASICA, prompts appear as in Figure 1.2 when you type the BASICA command. With BASICA, you can begin typing a program immediately.

Steps Involved in Writing Programs

Before discussing QBASIC and BASICA in depth, let us review the steps involved in writing programs and the techniques that should be used for creating well-designed programs.

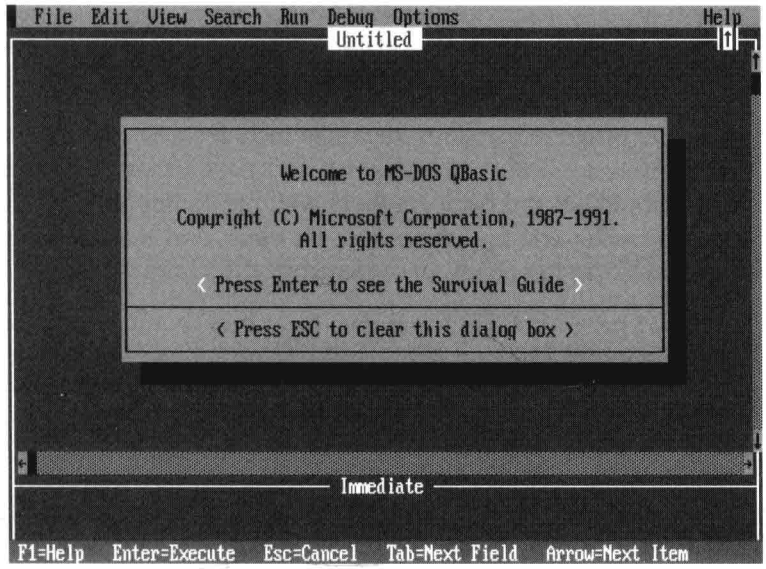


Figure 1.1
Initial QBASIC screen.

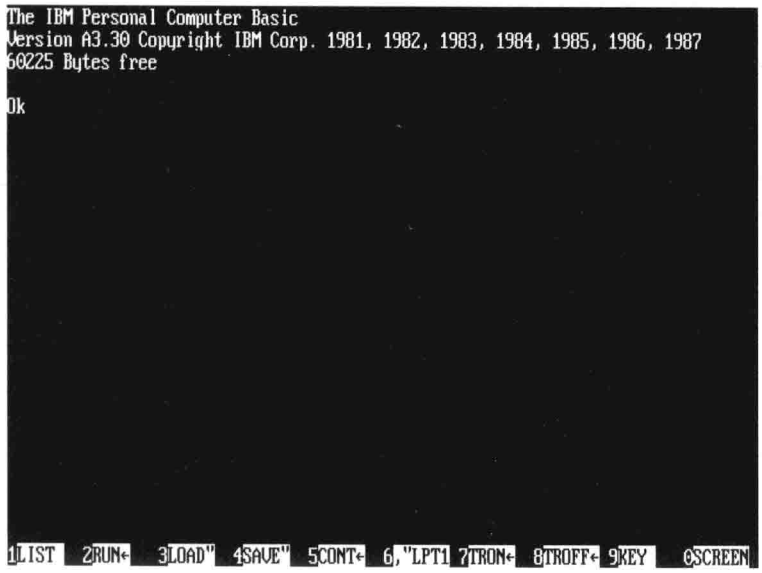


Figure 1.2
Initial BASICA screen.

1. Obtain the program requirements and specifications from a systems analyst or from the user. A programmer can interact with either, depending on the organization.
2. Plan the program using standard planning tools such as pseudocode and hierarchy charts.
3. Write the program and desk-check it before keying it in.
4. Key in the program and desk-check it again; watch out for typos.
5. Translate the program by means of either a BASIC compiler or interpreter. If

there are any rule violations or syntax errors, they will be displayed on the screen. Fix them and retranslate the program until there are no more syntax errors.

6. Run the program using test data that includes a sample of what the actual data is likely to look like. Using pencil and paper, manually determine the results you should get and compare them with the computer-produced results. If there are discrepancies, find the errors, fix them, and retranslate and rerun the program until all the results are correct.
7. Test the program with actual data to ensure that it runs smoothly in a normal operating environment.
8. Document the program so that users can run it without your intervention.

2

Techniques for Good Program Design

Before you begin to write actual programs in the BASIC language, take the time to review the following fundamental techniques for good program design.

Structured Programming

The most important technique for coding well-designed programs is called **structured programming.** Structured programming standardizes program design so that all programs, regardless of the language in which they are written, have a similar form. In general, structured programs are easier to read than nonstructured programs. They are also easier to debug and modify if changes are required at a later date. Moreover, they are easier to evaluate: programming managers and systems analysts are better able to assess programmers' skills and the quality of their programs.

For those of you who have had some previous programming experience, you may have encountered nonstructured techniques such as the frequent use of GOTOs, which is another term for a branch instruction. One major purpose of structured programming is to simplify debugging by reducing the number of entry and exit points (or "GOTOs") in a program. For that reason, structured programming is sometimes referred to as GOTO-less programming. Through the techniques of structured programming, the GOTO statement is avoided entirely. In BASIC, this means writing programs in which sequences are controlled by WHILE loops or some other logical control statement.

Modular Programming

Long and complex structured programs are sometimes subdivided into **modules**—also called **subroutines,** subprograms, or procedures. These are separate sets of instructions that accomplish distinct functions. Programs that are subdivided into modules are called modular programs; that is, subroutines, subprograms, or procedures are written as separate sections and are called into the main body of a program when they are needed.

Program modules are not only written separately, but they also are often tested independently. This makes it possible to segment a large or complex program into individual sections so that, if necessary, different programmers can code and debug these different sections. In summary, modular programs consist of individual sections that are executed under the control of a main module.

Top-Down Programming

Another important design technique that makes programs easier to read and more efficient is called **top-down programming.** Top-down programming implies that proper program design is best achieved by designing and coding major modules before minor ones. Thus, in a top-down program, the main modules are coded first, then intermediate modules are coded, and finally the minor ones are coded. By coding modules in this top-down manner, the organization or flow of the program is given primary attention.

The standardized top-down technique provides an effective complement to structured programming, thereby achieving efficient and effective program design.

In this text we use structured techniques in all our programs and avoid GOTOs entirely. In addition, we use a top-down approach in the more advanced programs so that you will learn to program in a manner that is widely accepted as a standard one.

3

A Sample BASIC Program

To understand the nature of BASIC, we will first look at a sample program that solves a typical business problem. This program will run with all versions of the BASIC language.

Definition of the Problem

A computer center of a company is assigned the task of calculating weekly wages (gross pay) for all nonsalaried personnel. The employee name, hourly rate, and number of hours worked are supplied as input for each employee, and the weekly wages figure is to be computed as follows:

$$\text{Weekly Wages} = \text{Hours Worked} \times \text{Hourly Rate}$$

Before processing can begin, the incoming data or input must be in a form that is "readable" by the computer. The input may be keyed in or read in from a disk or other storage medium.

Input Layout

Let us assume that the employee data will be keyed by means of a keyboard. As you will see later, the device used for entering the input does not really affect the program's logic.

Each employee's data consists of three fields called **variables**. A variable is a storage area that will contain data. We will name the three variables that will contain inputted data EMPNAME$, HOURS, and RATE. Later, when we discuss rules for forming variable names, you will see that the $ in EMPNAME$ is required when you are defining a variable that can contain alphabetic data.

Output Layout and Definition

For each EMPNAME$, HOURS, and RATE entered on the keyboard, the computer will display on a screen the employee's name and his or her weekly WAGES, which will be calculated as HOURS multiplied by RATE.