new brief edition

4TH EDITION

Turbo Pascal

Elliot B. Koffman
with Bruce Maxim

To our families, with much appreciation for their love and support—

Caryn, Richard, Deborah, and Robin Koffman

Norma Jean, Benjamin, and Katherine Maxim

Lynne Doran Cote, Sponsoring Editor
Jim Rigney, Senior Production Supervisor
Nancy Benjamin, Outside Production Coordinator
Joyce C. Weston, Text Designer
Joseph Vetere, Technical Art Consultant
Tech-Graphics, Illustrations
Trish Gordon, Manufacturing Manager
Sharon Elwell-Smizter Design, Cover Design
Peter M. Blaiwas, Cover Art Director

Library of Congress Cataloging-in-Publication Data

Koffman, Elliot B.
Turbo Pascal / Elliot B. Koffman with Bruce R. Maxim. —4th ed.
p. cm.
Abridged edition of author's earlier 4th ed. with same title.
Includes index.
ISBN 0-201-53920-9
1. Pascal (Computer program language) 2. Turbo Pascal (Computer file)
I. Maxim, Bruce R. II. Title.
QA76.73.P2K635 1993
005.26'2--dc20 92-38717
CIP

1 2 3 4 5 6 7 8 9 10 HA 979695949392

Preface

This is a textbook for a first course in problem solving and program design using the Pascal language and the Turbo Pascal Integrated Development Environment. It assumes no prior knowledge of computers or programming. High school algebra is sufficient mathematics background for most of the material in this book. A limited knowledge of discrete mathematics, however, is desirable for certain sections.

Our goal in revising this book has been to update the content and organization in accordance with the most recent computer science curricula recommendations. The last edition closely followed the recommendations of the ACM Computing Curricula Task Force for CS1[1] and CS2[2]. Many of the changes in this edition have been influenced by the recent report of the ACM/IEEE-CS Joint Curriculum Task Force[3].

This new edition provides more emphasis on problem solving, software engineering, abstraction, and computing theory. We also consolidated all information on decision statements into Chapter 4, and all information on loops into Chapter 5. To accomplish these changes, the first twelve chapters have been heavily revised.

Turbo Pascal Versions

Although the text uses screens from the latest version of Turbo Pascal (version 7.0), it can also be used with any earlier version of Turbo Pascal that supports units (versions 5.0 and later). The user screens will be different for earlier versions, but students should have no difficulty adapting the material in the text to an earlier environment. You will need Turbo Pascal 5.5 or later to utilize the chapter on object-oriented programming (Chapter 18), which appears in the expanded version of the book (see the discussion on Coverage of Advanced Topics).

The book can be used with either the MS-DOS or Microsoft Windows

1. Recommended Curriculum for CS1, Koffman, E., Miller, P., and Wardle, C., Communications of the ACM 27, 10 (Oct. 1984), 998–1001.

2. Recommended Curriculum for CS2, Koffman, E., Stemple, D., and Wardle, C., Communications of the ACM 28, 8 (Aug. 1985), 815–818.

3. Computing Curricula 1991, Report of the ACM/IEEE-CS Joint Curriculum Task Force, Tucker, A.B. (editor), 1991, ACM Press and IEEE Computer Society Press.

operating system. If you are using Turbo Pascal for Windows, you must insert the statement

```
uses WinCrt;
```

right after the program statement.

Problem Solving

The connection between good problem-solving skills and effective software development is established early in Chapter 1 with a new section that discusses the art and science of problem solving. Chapter 1 also introduces a methodology for software development based on the systems approach to problem solving consisting of five phases: specify the problem, analyze the problem, design the solution, implement the solution, test and verify the solution. The software development method is used in Chapter 2 to solve the first case study and is applied consistently to all the case studies in the text.

Chapter 3 continues the emphasis on problem solving by discussing top-down design, divide-and-conquer, solution by analogy, and generalizing a solution. An important section of this chapter demonstrates how a Pascal program can be derived by editing the documentation that results from systematically following the software development method.

Software Engineering

Many aspects of software engineering are covered in the book. Program style issues are discussed throughout in special displays. The concept of a program as a sequence of control structures is introduced early in Chapter 3. There are sections in several chapters that discuss algorithm tracing, using the Turbo Pascal debugger, and testing.

Chapter 9 is a new chapter on software engineering. This chapter discusses the system/software life cycle (SLC), prototyping, and programming teams. There is in-depth coverage of all phases of the SLC, including more discussion of informal techniques for program testing (e.g., glass-box versus black-box testing, integration testing, structured walkthroughs) and formal methods for program verification, including a discussion of loop invariants. This chapter also reviews procedural abstraction and introduces data abstraction. In this chapter, we introduce Turbo Pascal units and use unit `Crt` to create window-like interfaces. We also show how to write your own units to implement procedure libraries and abstract data types. Chapter 9 concludes with a discussion of professional ethics.

Procedural Abstraction

Although there is no universal agreement on when to introduce procedures and procedure parameters, most educators agree on the following points: procedures should be introduced as early as feasible, procedures should never process global data (side-effects), and the mastery of procedure parameters is

challenging. The approach taken in the text is to discuss the importance of program modularization and reusability in Chapter 3 by introducing structure charts, procedures without parameters, and the standard functions of Pascal. Sections 3.5 and 3.6 motivate the use of procedures as program-building blocks by showing some applications of procedures without parameters (e.g., displaying long lists of user instructions, drawing diagrams). Section 3.7 discusses the need for parameters and the limitations of procedures without parameters, thereby providing a foundation for the later study of parameters. Section 3.8 shows how to use Pascal's predefined functions.

Chapter 6 completes the study of procedures and functions, covering all aspects of parameter lists. The chapter begins by discussing procedures with only value parameters, then user-defined functions with value parameters, and, finally, procedures with both value and variable parameters. An optional section at the end of the chapter introduces recursive functions.

Some instructors prefer to cover procedures with and without parameters together. You can easily rearrange the sequence of topic coverage to do this. If you want to wait until Chapter 6 to cover procedures with and without parameters, you can defer sections 3.5–3.7 until then. Conversely, if you want to cover procedure parameters earlier, you can cover section 6.1 (value parameters) right after Chapter 3. You can cover sections 6.2–6.4 (user-defined functions, variable parameters, syntax of parameter lists) after completing the first two sections of Chapter 4 (Boolean expressions and the `if` statement).

Data Abstraction

The software engineering chapter (Chapter 9) introduces data abstraction, providing the first example of an abstract data type (ADT). We introduce enumerated data types and show how to encapsulate a data type and its operators as a Turbo Pascal unit. Data abstraction and ADTs are used throughout the remainder of the text. The ADTs appearing in Chapters 11 and 12 have been extensively revised and improved.

In the expanded version of the text, Chapter 18 discusses object-oriented programming. We compare and contrast the use of ADTs and objects and discuss the important concepts of inheritance and polymorphism, which are central to the object-oriented paradigm.

Consolidation of Decision and Loop Statements

Comments from many previous users of the textbook indicated that the majority of instructors prefer to discuss both Pascal control structures for selection (`if` and `case`) at the same time. For this reason, Chapter 4 has been revised to cover the Boolean data type and the `if` and `case` control structures. Similarly, all three control structures for repetition (`while`, `for`, and `repeat`) are covered together in Chapter 5.

The coverage of files has also been consolidated. The essentials of file usage are discussed in Chapter 2, so that student programs can read data files pre-

pared by the instructor. Chapter 8 provides complete coverage of text files, and Chapter 15 of the expanded version covers binary files.

Interviews with Computer Scientists

A new feature of this edition is a collection of interviews with several notable computer scientists (e.g., Peter Denning, Patrick Winston, Adele Goldberg, Philippe Kahn, C. J. Date, and others), which are placed throughout the text. These interviews alert beginning students to the breadth of the subject area, providing them with a description of issues of concern in several fields of computer science (e.g., artificial intelligence, operating systems, databases, user interfaces) and some idea of the background preparation needed for success in these fields.

Increased Coverage of Theoretical Concepts

As recommended by the curriculum committee report, there is increased coverage of theoretical topics. Chapter 7 introduces numerical computation and iterative approximations, including Newton's method. Chapter 9 provides a discussion of program verification, focusing on assertions and loop invariants. Chapter 10 introduces searching and sorting an array, followed by a discussion of algorithm analysis and big-O notation.

Pedagogical Features

We employ several pedagogical features to enhance the usefulness of this book as a teaching tool. Some of these features are discussed below.

End-of-Section Exercises: Most sections end with a number of self-check exercises. These include exercises that require analysis of program fragments as well as short programming exercises. Answers to odd-numbered self-check exercises appear at the back of the book; answers to the rest of the exercises are provided in the instructor's manual.

End-of-Chapter Exercises: Each chapter ends with a set of quick-check exercises with answers. There are also chapter review exercises whose solutions appear in the instructor's manual.

End-of-Chapter Projects: Approximately one-third of the programming projects are new to this edition. Most chapters have one or two special programming project pairs where the second project in the pair requires a modification to the solution of the first project in the pair. The program disk (described below) contains a solution to the first project in each pair, which students can modify to solve the follow-up project. All project solutions appear in the instructor's manual.

Examples and Case Studies: The book contains a large number and variety of programming examples. Whenever possible, examples contain complete programs or procedures rather than incomplete program fragments. Each chapter

contains one or more substantial case studies that are solved following the software development method. This edition contains several new case studies.

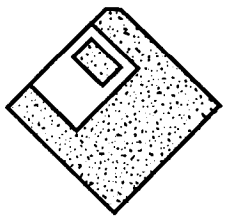
Syntax Display Boxes: The syntax displays describe the syntax and semantics of each new Pascal feature and provide examples. There are also several syntax diagrams in the body of the text; Appendix C contains a complete collection of syntax diagrams.

Program Style Displays: The program style displays discuss issues of good programming style.

Error Discussions and Chapter Review: Each chapter ends with a section that discusses common programming errors. A chapter review includes a table of new Pascal constructs.

Program Disk

There is a program disk that contains all of the programs, procedures, functions, and ADTs introduced in the book. For case studies, all procedures and functions are incorporated in a single program file. There are also solutions to selected programming projects on the disk. These solutions serve as the starting point for follow-up on projects. Programs on the disk can be loaded into the Turbo Pascal environment. The icon



Directory: CHAP2
File: METRIC.PAS

appears alongside each program module or project indicating the name of the disk directory (e.g., CHAP2) and file (e.g., METRIC.PAS) containing the program.

Laboratory Manual

A new feature of this edition is a laboratory manual, including a program disk. This manual provides support for a programming laboratory based on the book. A number of laboratory exercises test a student's understanding of new concepts and provide additional practice in their application.

Coverage of Advanced Topics

The material in Chapters 1 through 12 will normally be covered in the first semester of a course in programming methods. The expanded version of the

book contains six additional chapters, which cover advanced topics normally studied in the second semester. The expanded version may be used as a reference for students continuing their study of computer science, and, in some cases, may be used as the text for a second-semester course. There is certainly sufficient material for a two-quarter sequence. Advanced topics covered include:

recursion (Chapter 13)
sets and strings (Chapter 14)
binary files (Chapter 15)
pointers and linked lists (Chapter 16)
stacks, queues, and trees (Chapter 17)
object-oriented programming (Chapter 18)

Faculty should order the expanded version of the book if they feel they will be covering more than the basic content in their first course, or if they plan to use the book for more than one quarter or semester. Also, if a relatively large percentage of students in the first course continue their study of computer science, it would be desirable to order the expanded version to provide students with an alternate reference to these topics.

Appendixes, Supplements, and Ordering Information

Separate appendixes cover the Turbo Pascal environment, language elements, compiler directives, syntax diagrams, and the ASCII code.

An instructor's manual is available for this edition. Other supplements include transparency masters and a lab manual. Use the reference numbers below to order these supplements from your Addison-Wesley sales representative.

Lab Manual with 3½" Disk: 0-201-51583-0
Instructor's Manual: 0-201-55812-2
Transparency Masters: 0-201-55813-0
Program Disk (if purchased separately): 0-201-55814-9

There are several ordering options available:

Text (Chapters 1-12): 0-201-53920-9
Text (Chapters 1-12) with 3½" Program Disk: 0-201-54212-9
Expanded Text (Chapters 1-18): 0-201-55811-4
Expanded Text (Chapters 1-18) with 3½" Program Disk: 0-201-52442-2

Acknowledgments

The principal reviewers were most essential in suggesting improvements and finding errors. They include: Pierre Balthazard, University of Arizona; Taylor Binkley, Georgia State University; Deborah Byrum, Texas A & M University; Robert Christiansen, University of Iowa; Edwin Ellis, Mississippi State University; John Goda, Georgia Institute of Technology, Anil Mehra, Colorado State University; Anne W. Oney, De Anza College; Keith Pierce, University of Min-

nesota; James C. Pleasant, East Tennessee State University; Michael C. Stinson, Central Michigan University; and Robert Strader, Stephen F. Austin State University. Besides reviewing the text, Professor Pleasant helped with the section on software verification.

We are also grateful to the many teachers who participated in telephone interviews or completed course surveys for Addison-Wesley's market research department. The information this research provided helped to shape the book's organization and pedagogy. They include: Thomas Ahlborn, West Chester University; Dean R. Andrews, Texas State Technical College; Daniel J. Barrett, University of Massachusetts; John M. Barton, Freed-Hardeman University; George A. Benjamin, Muhlenberg College; Taylor Binkley, Georgia State University; Susan Bonzi, Syracuse University; Bill Boyd, Rhodes College; Linda D. Brinkerhoff, Erie Community College; John F. Buck, Indiana University; Allen R. Burns, Rutgers University; Mae M. Carpenter, Georgia College; Kan V. Chandras, Fort Valley State College; Darrah Chavey, Beloit College; Thomas G. Clarke, North Carolina A & T State University; Edwin Ellis, Mississippi State University; Michael Erickson, Wichita State University; Susan Gauch, Northeastern University; Jarrell C. Grout, Stephen F. Austin State University; Stan Gurak, University of San Diego; Arthur Jackman, Dean Junior College; Ron Johnson, Evangel College; Mike Liljegren, Illinois College; Slawomir J. Marcinkowski, Syracuse University; Tom McQueary, Tarrant County Junior College; William Moy, University of Wisconsin; Michael G. Murphy, University of Houston; David A. Nelson, Muhlenberg College; James Nolen, Baylor University; James L. Noyes, Wittenberg University; Sue Pilgreen, McNeese State University; Cyndi Rader, Wright State University; Ed Rang, University of Wisconsin; Brian Ridgely, Alma College; Patricia Shelton, North Carolina A & T State University; Robert G. Strader, Stephen F. Austin State University; Vicci Varner, University of Texas; Leila L. Wallace, Geneva College; and Stephen Weiss, University of North Carolina.

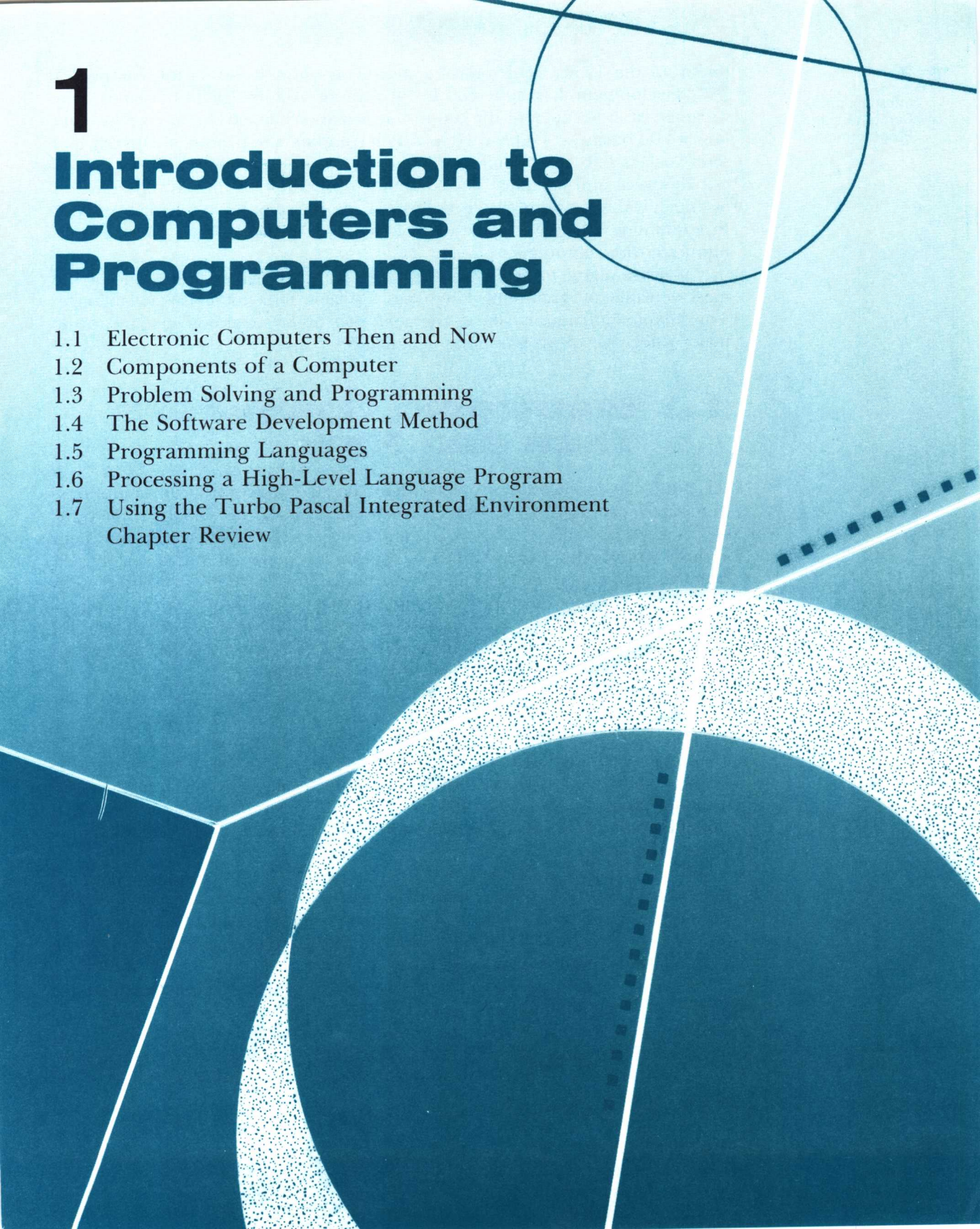
The personnel at Addison-Wesley responsible for the production of this book worked diligently to meet a very demanding schedule. Our editor, Lynne Doran Cote, was closely involved in all phases of this project. Ably assisted by Andrea Danese, Lynne did an excellent job of coordinating the writing and reviewing process and trying to keep us on a very tight schedule. Jim Rigney supervised the production of the book, while Nancy Benjamin coordinated the conversion of the manuscript to a finished book. We are grateful to all of them for their fine work.

Philadelphia, PA E.B.K.
Dearborn, MI B.R.M.

Contents

1

Introduction to Computers and Programming



From the 1940s until today—a period of only 50 years—the computer's development has spurred the growth of technology into realms only dreamed of at the turn of the century. It has also changed the way we live and how we do business. Today we depend on computers to process our paychecks, send rockets into space, build cars and machines of all types, and help us do our shopping and banking. The computer program's role in this technology is essential; without a list of instructions to follow, the computer is virtually useless. Programming languages allow us to write those programs and thus to communicate with computers.

You are about to begin the study of computer science using one of the most versatile programming languages available today: the Pascal language. This chapter introduces you to the computer and its components and to the major categories of programming languages.

1.1 Electronic Computers Then and Now

It is difficult to live in today's world without having some contact with computers. Computers are used to provide instructional material in schools, print transcripts, send out bills, reserve airline and concert tickets, play games, and help authors write books. Several kinds of computers cooperate in dispensing cash from an automatic teller machine; "embedded" or "hidden" computers help control the ignition, the fuel system, and the transmission of modern automobiles; at the supermarket a computer device reads the bar codes on packages to total your purchases and help manage the store's inventory. Even microwave ovens have special-purpose computers built into them.

Computers were not always so pervasive in our society. Just a short time ago, computers were fairly mysterious devices that only a small percentage of the population knew much about. Computer know-how spread when advances in *solid-state electronics* led to cuts in the size and the cost of electronic computers. In the mid-1970s, a computer with the computational power of one of today's personal computers would have filled a 9-by-12-foot room and cost $100,000. Today an equivalent personal computer (Fig. 1.1) costs less than $3,000 and sits on a desktop.

If we take the literal definition for *computer* as "a device for counting or computing," then we could consider the abacus to be the first computer. The first electronic digital computer was designed in the late 1930s by Dr. John Atanasoff at Iowa State University. Atanasoff designed his computer to perform mathematical computations for graduate students.

The first large-scale, general-purpose electronic digital computer, called the ENIAC (Electronic Numeric Integrator And Computer), was built in 1946 at the University of Pennsylvania. Its design was funded by the U.S. Army, and it was used to compute ballistics tables, predict the weather, and make atomic energy calculations. The ENIAC weighed 30 tons and occupied a 30-by-50-foot space (Fig. 1.2).

Figure 1.1 IBM Personal Computer with Mouse

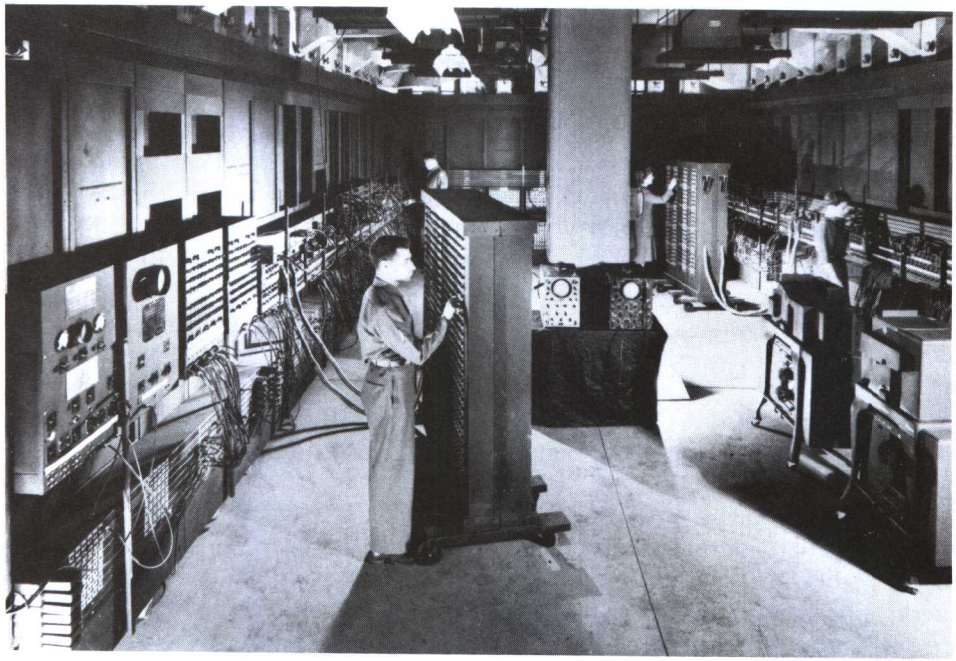


Figure 1.2 The ENIAC Computer (Photo Courtesy of Unisys Corporation)

Although we are often led to believe otherwise, computers cannot reason as we do. Basically, computers are devices that perform computations at incredible speeds (more than one million operations per second) and with great accuracy. However, to accomplish anything useful, a computer must be *programmed,* that is, given a sequence of explicit instructions (a *program*) to perform.

To program the ENIAC, engineers had to connect hundreds of wires and arrange thousands of switches in a certain way. In 1946 Dr. John von Neumann, of Princeton University, proposed the concept of a *stored-program computer*: a program stored in computer memory rather than set by wires and switches. Von Neumann knew programmers could easily change the contents of computer memory, so he reasoned that the stored-program concept would greatly simplify programming a computer. Von Neumann's design was a success and is the basis of the digital computer as we know it today.

Brief History of Computing

Table 1.1 lists some of the important milestones along the path from the abacus to modern-day computers and programming languages. The entries before 1890 list some of the earlier attempts to develop mechanical computing devices. In 1890 the first special-purpose computer that used electronic sensors was designed; this invention eventually led to the formation of the computer-industry giant called IBM (International Business Machines).

As we look down the table from 1939 onward, we see a variety of new computers introduced. The computers listed before 1975 were all very large general-purpose computers, called *mainframes.* The computers listed after 1975 are all smaller computers.

A number of milestones in the development of programming languages and environments are also listed in Table 1.1, including Fortran (1957), CTSS (1965), Pascal (1971), VisiCalc (1978), Turbo Pascal (1983), and Windows (1985).

We often use the term *first generation* to refer to electronic computers that used vacuum tubes (1939–1958). The *second generation* began in 1958 with the changeover to transistors. The *third generation* began in 1964 with the introduction of integrated circuits. The *fourth generation* began in 1975 with the advent of large-scale integration.

Categories of Computers

Modern-day computers are classified according to their size and performance. The three major categories of computers are microcomputers, minicomputers, and mainframes.

Many of you have seen or used *microcomputers,* such as the IBM PC (see Fig. 1.1). Microcomputers are also called *personal computers* or *desktop computers* because they are used by one person at a time and are small enough to fit on a desk. The smallest general-purpose microcomputers are often called *laptops* because they are small enough to fit into a briefcase and are often used on one's lap. The largest microcomputers, called *workstations* (Fig. 1.3), are commonly