INTRODUCTION TO COMPUTERS,
STRUCTURED PROGRAMMING, AND APPLICATIONS

Module
C

Computers and Systems

C. WILLIAM GEAR

INTRODUCTION TO COMPUTERS, STRUCTURED PROGRAMMING, AND APPLICATIONS

Module

C

Computers and Systems including General Introduction

C. WILLIAM GEAR
University of Illinois
Urbana, Illinois

SRA

SCIENCE RESEARCH ASSOCIATES, INC.
Chicago, Palo Alto, Toronto, Henley-on-Thames, Sydney, Paris, Stuttgart
A Subsidiary of IBM

Compositor	Advanced Typesetting Services
Acquisition Editor	Robert L. Safran
Project Editor	Jay Schauer
Special Editorial Assistance	Stephen B. Chernicoff
Text Design	Judy Olson
Cover Design	Michael Rogondino

ACKNOWLEDGMENTS

Figure C4.2a, courtesy of Control Data Corporation; figures C4.3, C4.4, C4.7, C4.11, C4.12, courtesy of IBM; figure C4.5, courtesy of Teletype Corporation; figure C4.6, courtesy NCR; figure C4.8, courtesy of California Computer Products; figures C6.1, C6.2, courtesy Intel Corporation.

LIBRARY OF CONGRESS CATALOGING IN PUBLICATION DATA

Gear, Charles William.
Computers and systems.

(His Introduction to computers, structured programming, and applications)
Includes index.
1. Electronic digital computers. I. Title.
II. Series: Gear, Charles William. Introduction to computers, structured programming, and applications.
QA76.5.G362 001.6'4 77-25547
ISBN 0-574-21191-8

10 9 8 7 6 5 4 3

INTRODUCTION TO COMPUTERS, STRUCTURED PROGRAMMING, AND APPLICATIONS

Module

C

Computers and Systems including General Introduction

Preface to This Series

The origins of this material lie in the earlier text *Introduction to Computer Science,* which taught programming and problem solving in a language-independent way. The present text, a thorough revision of the earlier material, seeks the same goal, but places strong emphasis on the structured, top-down style of problem solving and program development, which has been found to produce better programs that can be written and debugged more quickly. Programs are presented in an informal "pseudolanguage" that can easily be transliterated to common procedure-oriented languages. Separate language manuals, which discuss the details of a number of widely used languages, are available with the text. Each of these language manuals shows how the programming principles developed in the text can be implemented in a particular language, and gives versions in that language of many of the example programs from the text.

An informal language is used in this text in preference to the earlier book's flowcharts because flowcharts can be used in an unstructured way, and can encourage sloppy thinking and sloppy programming. Despite the recent popularity of "structured" flowcharts, no uniform format for their use has been accepted, so they are seldom used in this text. The informal language is written in such a way, however, that programs can easily be converted into a version of structured flowcharts by drawing the appropriate boxes around sections of code (see Chapter G3).

The text consists of a general introduction and three modules: Module C, Computers and Systems; Module A, Algorithms and Applications; and Module P, Programming and Languages. This structure enables the student or instructor to choose both the material to be covered and its ordering with great flexibility. After the General Introduction, Module P can be covered in order or some of its chapters can be postponed. At the same time, applications from Module A can be used

to illustrate the material on programming and to develop problem-solving skills. Supplementary material from Module C can be used at any time. For example, some instructors might prefer to cover some material on computer organization and machine language before getting involved in higher-level languages, by teaching Chapters C1 and C2 before beginning Module P. Alternatively, these chapters may be delayed until later, or even omitted entirely. To help the instructor select an order of instruction, a diagram appears at the beginning of each module, showing specific prerequisites for each chapter. The general introduction should be treated as a prerequisite to any of the other modules.

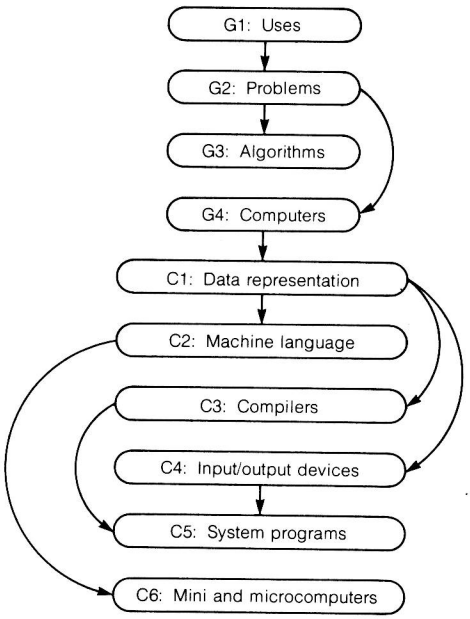
Each volume includes an appendix giving solutions to selected problems from the ends of the chapters. Problems for which an answer is given are marked in the problem sets with an asterisk (*); a dagger (†) indicates that a partial answer is given.

The language manuals are designed specifically to complement this text. As general programming material is developed here, the details for a particular language are developed in the corresponding chapter in the language manual. The reader should read the two chapters at about the same time, first the chapter in the text for the basic principles, then the corresponding chapter in the language manual for their specific implementation. If the instructor wants to teach a second language, the student is well prepared after learning one language in this way. (The two languages should not be taught in parallel, however, but serially.) The objective in teaching a specific language this way is to prepare the student to learn other languages quickly and easily.

Several versions of Module A (Algorithms and Applications) are available, so that the instructor can select the one that best serves the needs of the students. A complete set of modules (C, A, P, and a language manual) can provide enough material for a two-semester sequence if all the applications are covered. (A second language can be taught in the second semester, if desired, while developing the applications.) Alternatively, some of the applications can be selected along with material from Modules C and P to form a one-semester course.

Development of this material would not have been possible without the capable editorial assistance of Stephen B. Chernicoff. Many people at SRA, including Jay Schauer and Bob Safran provided help and encouragement. I am grateful for the valuable comments and suggestions of those who reviewed this text—Robert Cannon, University of South Carolina, J. Flaherty, Renssalaer Polytechnic Institute, Olin G. Johnson, University of Houston, Larry D. Wittie, State University of New York—and especially for the help of Marilyn Bohl, who reviewed and constructively criticized many versions of both the earlier text and the present one.

C. William Gear



Prerequisite structure for Module C

Contents

General Introduction

Computer science is a discipline concerned with the capabilities of computers, the types of problems they can be used to solve, and the ways in which problems should be approached. You do not have to be a computer scientist to use a computer: the applications of computers extend from science and engineering to business and the humanities. The aim of this text is to introduce you to the basic principles of computers and their use.

Our subject can be broken down into three related areas of study: how the computer works, how to solve problems with it, and how to communicate the method of solution to it. The text comprises three main modules corresponding to these areas of study. The General Introduction offers an overview of the three areas and their interrelations, and is intended to provide the background needed for the study of any of the three individual areas.

Module C (Computers and Systems) deals with the internal organization and operation of computers and computer systems. How much of this material should be covered will depend on the needs of the individual student, although some acquaintance with the computer's organization is essential for an understanding of its efficient use. The type of problems being discussed and the programming language being used will determine which chapters of Module C should be studied.

Module A (Algorithms and Applications) focuses on computer applications and methods of problem solution. There are many types of problems, each requiring different techniques for solution. Although they have common underlying ideas, you may wish to concentrate on those problems that interest you. Once you have become a competent programmer, you should be able to solve a broad range of problems.

Module P (Programming and Languages) describes programming itself: the process of breaking a job into a sequence of simple steps that can be executed by a computer. Programs in the text will be written in an informal "pseudolanguage" embodying the common features of most typical programming languages. Companion language manuals show how these same programs are actually written in a variety of programming languages.

The computer scientist seeks general principles as they apply to whole classes of problems; the application programmer is interested only in finding the best solution to a given problem in a particular application area. Both must be able to program a wide spectrum of problems effectively. Some people regard programming as an art, others argue that it is a science. The label is not important—what *is* important is to develop your programming skills so that you can understand computers and apply them to your own problems.

Chapter

G1

Uses of Computers

Computers are used in so many different ways in modern life, that they are frequently misnamed "electronic brains." In fact, all these different applications use the same basic principles, and all must first be analyzed as problems *by a person* before the computer can be used. For example, a computer may be used to perform routine work of a repetitive nature, such as maintaining the bank accounts of all the customers of a bank. Previously this job was done by manual labor. Bookkeepers were taught the rules to be followed for a typical account, and then they applied them to each account. Now the same task can be performed by a computer, but the rules must first be established by a person. This is an example of the use of a computer to save human drudgery in processing repetitive information (see Figure G1.1a).

Computers are also used to control systems in environments where it is not feasible for humans to act, as, for example, in unmanned space flight. A computer may be organized to respond in predetermined ways to measurements made by on-board instruments, such as radar, and to signals sent from ground stations. It may also compute the position and velocity of the spacecraft in order to find out where the craft is heading in relation to the desired destination. Using the information available, the computer can send signals to the spacecraft's control systems to keep the craft on the desired course and to perform the planned maneuvers. However, notice again that all the operations have to be planned ahead by people. A person has to think out the response to each combination of circumstances and organize the computer to produce those responses. Thus a person might decide that the braking rockets should be fired with a thrust proportional directly to the velocity and inversely to the distance from the target. When the computer is properly prepared, it can control

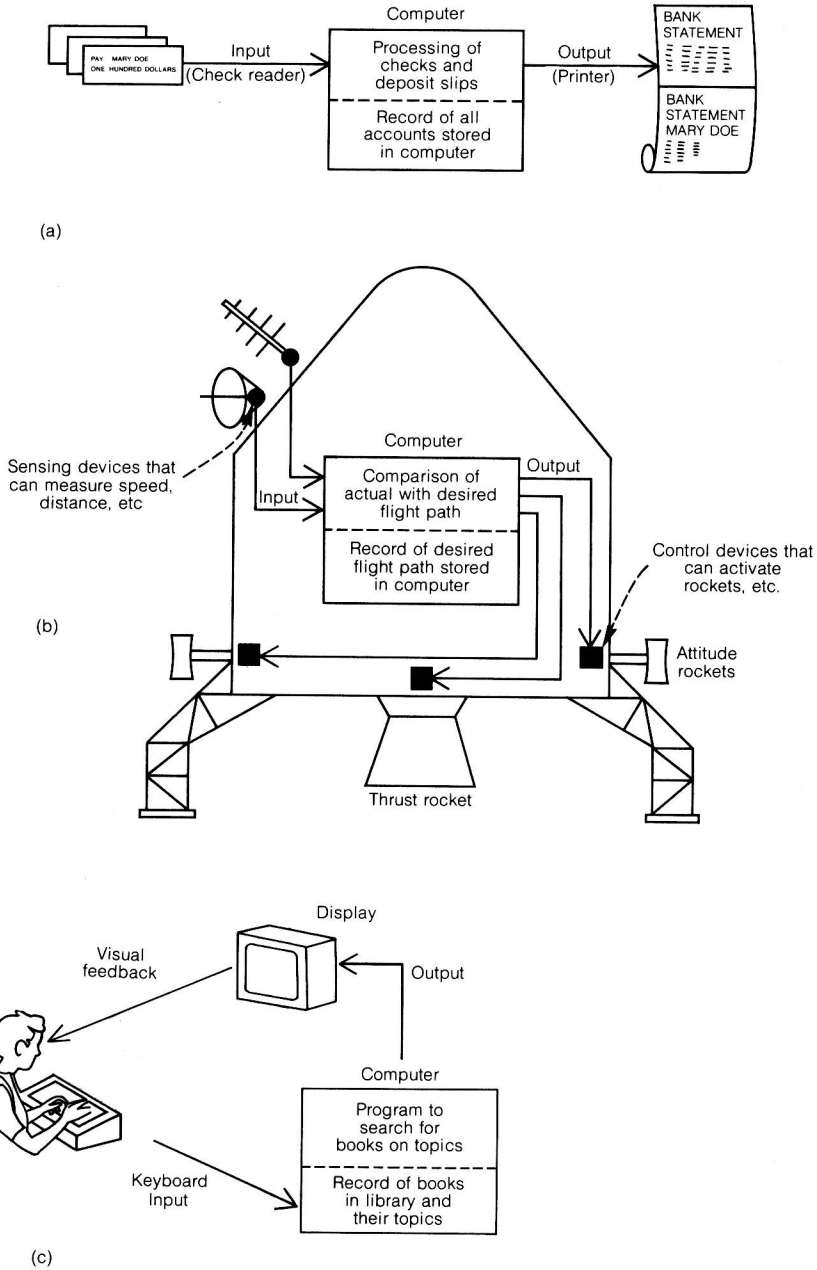


Figure G1.1 Three uses of a computer
(a) Processing of a repetitive kind—Bank deposits and withdrawals
(b) Control of complex tasks—Guiding a spacecraft
(c) Help in problem solving—Information retrieval

the braking rockets, but the decision about *how* they are fired is made by a person while planning the program. This type of computer use is shown in Figure G1.1b.

A third important area of computer application is to assist people in solving problems that are beyond human capabilities. For example, a computer can be used to perform long sequences of computations that could never be performed by people because of human slowness and proneness to error. Such problems commonly arise in mathematics or engineering when computations can only be performed *sequentially*—that is, when the results of one calculation must be known before the next can be performed. Manual calculation of the stresses in a modern airplane wing, for example, would be out of the question. Although the designer of the plane makes the basic decisions about the style of the wing and how the stresses are to be analyzed, it is the computer that makes it possible to perform the analysis in a practical length of time.

Calculations of this type are different from those that can be done in *parallel*. For example, if a bank attracts more customers, it will have to handle more transactions each day. It can make this possible either by getting more bookkeepers or by getting a computer. More bookkeepers can do the job because each bookkeeper can be working on a different transaction in parallel. But hiring more people would not make it possible to analyze a larger airplane wing than can be analyzed by one person, because a second person could not start calculating until the first had finished.

Another example of computer aid to limited human powers is in *information retrieval*. A researcher searching through a library may be aided by a computerized information-retrieval system that can examine all the records in the library and locate those that contain references to the topic of interest. In this application, the computer must be organized, before the search is actually performed, to search through the data in a particular order and perform the appropriate sequence of comparisons. The use of a computer to aid people in problem solving is shown in Figure G1.1c. Here we see that the action of the user may be influenced by the output from the computer—the results of the calculations are displayed to the user immediately, so that the input can be modified if necessary. If the book sought is not present, the user can search for alternative references. If the calculations show that a design is unacceptable, the user can change the design. This mode of computer use is called *interactive*, and is distinguished from *batch* use, in which the user leaves a job at the computer center to be processed and gets the output later, usually in printed form.

Although the *amount* of work performed in these examples by computers and people working together is much greater than could be performed by people alone, the *type* of work could be performed, in

principle, by people. In any case, the work must be planned and organized by people. These four example applications—bank record keeping, control of unmanned spacecraft, execution of large-scale calculations, and library search—all contain two common elements that must be considered in any problem we attempt to solve by computer.

The first element of a problem is its *control structure.* Various actions must be taken in response to various conditions. Thus in the bank application, one control action may be to send a nasty letter to the customer when a check is deducted from an account with insufficient funds to cover it. The spacecraft application consists mainly of control actions. As new data on position, velocity, and other conditions is received, the spacecraft's propulsion systems must be controlled to perform the desired maneuvers. In the calculation of wing stresses, control considerations are minor, although certain actions may be required if the stresses exceed limits previously set by the human designer. The library search may require a series of control actions that depend on the presence or absence of certain topic titles in the records.

The second element of a problem is the *data structure.* In the information-retrieval problem in particular, the structure of the data is the primary factor that must be considered in solving the problem. The desired information is already present in the stored data: the researcher needs to have it presented in a different form. Thus, in one sense, the problem of finding the telephone number of Mr. X. Smith of 2 Broadway, New York, is solved by handing the questioner a current copy of the New York telephone book—but the questioner wants the data organized in a different way, so that the required number can be read immediately. The questioner would be satisfied if we replied with the book open to the correct page, with the required number underlined clearly. We would have solved the problem by reorganizing the data. When we retrieve the information requested from a library, all we have done is reorganize the data and present it differently. In the sense that "two plus two" and "four" are different ways of saying, or presenting, the same information, all we are doing when we perform numerical computations is reorganizing the data—that is, changing its structure.

When we consider ways to solve a problem, we must examine the structure of the data initially available and decide on the structure of the data to be presented as the answer. This information may be completely specified in the description of the problem, or some of it may be left for the person solving the problem to decide. For example, if a table of numbers is to be presented, we may have the freedom to arrange those numbers in a column or in a row. If the table is large and has several sets of data to be placed in columns or rows, the table arrangement may affect the organization of the computer program that is written to solve the problem.

We must also examine the control actions that will be required and make sure all possibilities have been considered. What, for example, should the computer do if, in searching for a given topic in the library, it finds no references at all? Certainly it should not keep on searching forever; hence, when a person prepares the computer to perform this task, the action to be taken in this case must be considered.

Figure G1.1 also illustrates another very important feature of computers—their *input/output (I/O) mechanisms*. The computer must be able to communicate with its environment, to find out what to do and to return the results. There are many different forms of input and output. In Figure G1.1a a *check reader* and a *printer* are used. Today, all checks are encoded magnetically with the account number and other information, so they can be read by a computer. Before this development, the information would have been entered onto a *punched card* that could be read by a computer. Figure G1.1b shows *on-line* data acquisition and control. Figure G1.1c illustrates the use of an *interactive terminal*: a person types data at a keyboard connected to the computer and sees the results generated by the computer on a display screen.

Chapter C4 discusses a number of common input/output devices. Although these I/O units may appear very different, at the programming level they are all essentially equivalent. Input is received by the computer from the input device a character at a time, although in most languages we deal with a whole *string* of characters at a time: all the characters on one typed line, or from one check, or generated by a single sense device at a particular time. We will say that the computer *reads* a line of information, and will make no distinction among different forms of input. Similarly, on output, the computer transmits a string of characters to the output device. This string could be a line for a printer or display, or a set of commands to a control device such as a rocket engine. When we program, we will not have to know anything about the output device except the format of the string of characters to be transmitted, so we will not distinguish among different output devices.

Problems

***1.** Try to describe very briefly three applications of computers. The first should be of the routine type that could be done by people. The second should be of the type that is impossible for people to perform because of the limitations of human capabilities. The third should be of the type in which the computer enhances human capabilities.

***2.** Describe some aspects of the data and the control structures for each of your three problems.

Chapter

G2

Problem Solving

It is usually not expedient to use a computer to solve only one problem; rather, if a program is written, it should be capable of directing the computer to solve many similar problems. Thus it would not be worthwhile to write a program for the information-retrieval problem to look for just one book, but it would be valuable if many millions of books were to be referenced over a period of time. Similarly, it would not be practical to write a program to process only one bank transaction; such a program would be justified economically only if almost all transactions could be processed automatically. Therefore, when we write a computer program, we write it to solve a family of problems, not just one. We must determine the nature of this family of problems—the types of transactions to be permitted, the allowable size of the numbers, and so forth—and then prepare a program that will handle any of the allowed cases. We will say that we haved *solved the problem* when we have written such a program. (Handling a specific case of the data—for example, processing a single bank transaction—is solving a specific *instance* of the problem.)

A program for the computer consists of a sequence of operations. It contains only those operations that the computer can perform. *Problem solving* is the process of expressing the solution of complex problems in terms of the simple operations "understood" by the computer.

In order to solve a problem by computer, whether the problem is to control a spacecraft or to compute the deflection of a bridge when a train crosses it, we must pass through certain stages. These are:

1. *Formulate the problem precisely.* State all assumptions clearly and specify the actions to be taken in the event of any expected contingencies. (It is difficult to achieve the necessary precision when the problem is first stated. Usually, later stages of the process reveal