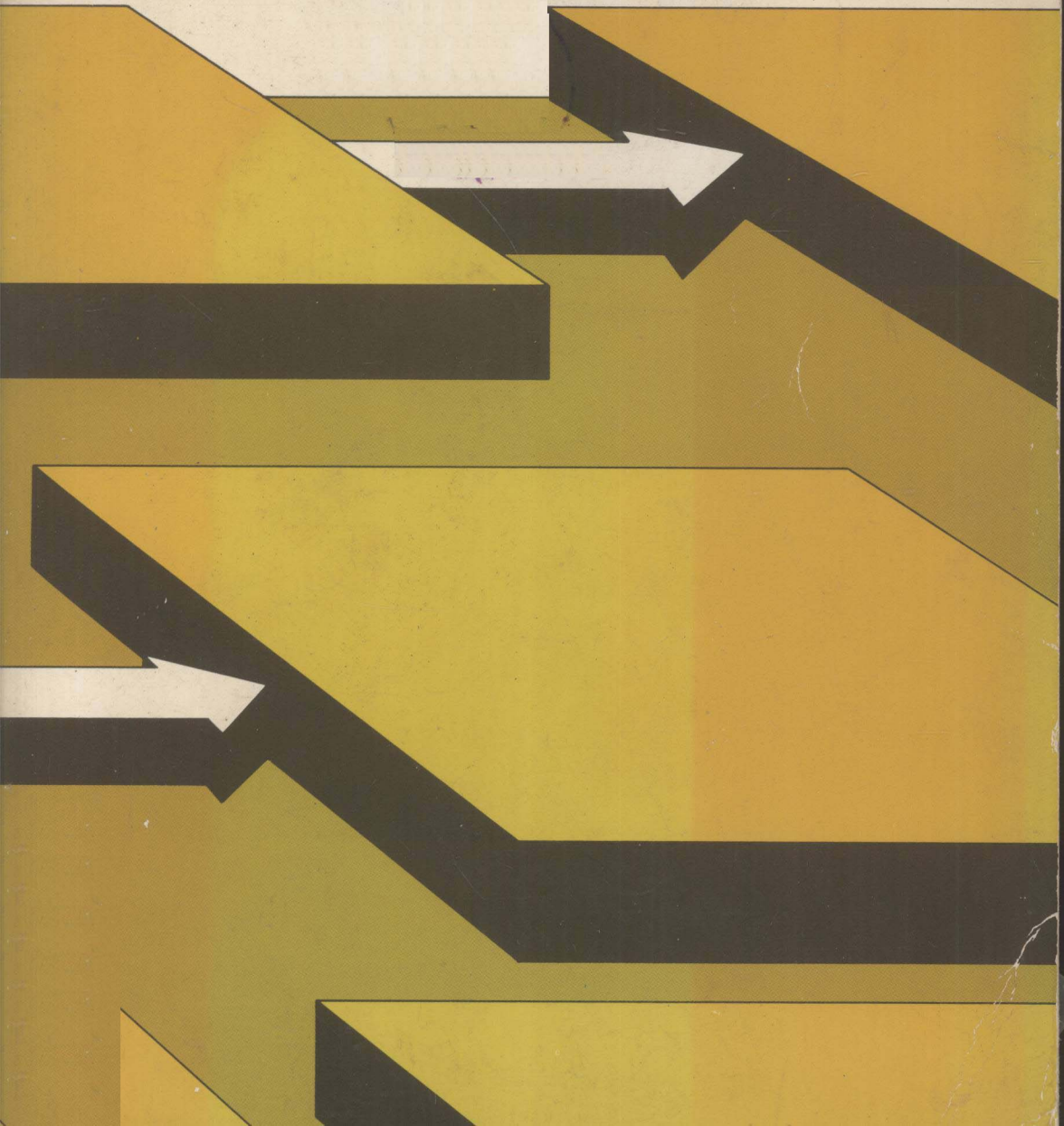
INTRODUCTION TO COMPUTERS, STRUCTURED PROGRAMMING, AND APPLICATIONS

Module
A

Applications and Algorithms in Business

FRANCES G. GUSTAVSON AND C. WILLIAM GEAR



INTRODUCTION TO COMPUTERS, STRUCTURED PROGRAMMING, AND APPLICATIONS

Module

A

Applications and Algorithms in Business

FRANCES G. GUSTAVSON
Pace University
Pleasantville, New York

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

LIBRARY OF CONGRESS CATALOGING IN PUBLICATION DATA

Gustavson, Frances G
Applications and algorithms in business.

(Introduction to computers, structured programming, and applications)
Includes index.
1. Business—Data processing. 2. Algorithms.
I. Gear, Charles William, joint author. II. Title.
III. Series.
HF5548.2.G87 658'.05'4 78-6091
ISBN 0-574-21190-X

10 9 8 7 6 5 4 3 2 1

INTRODUCTION TO COMPUTERS, STRUCTURED PROGRAMMING, AND APPLICATIONS

Module

A

Applications and Algorithms in Business

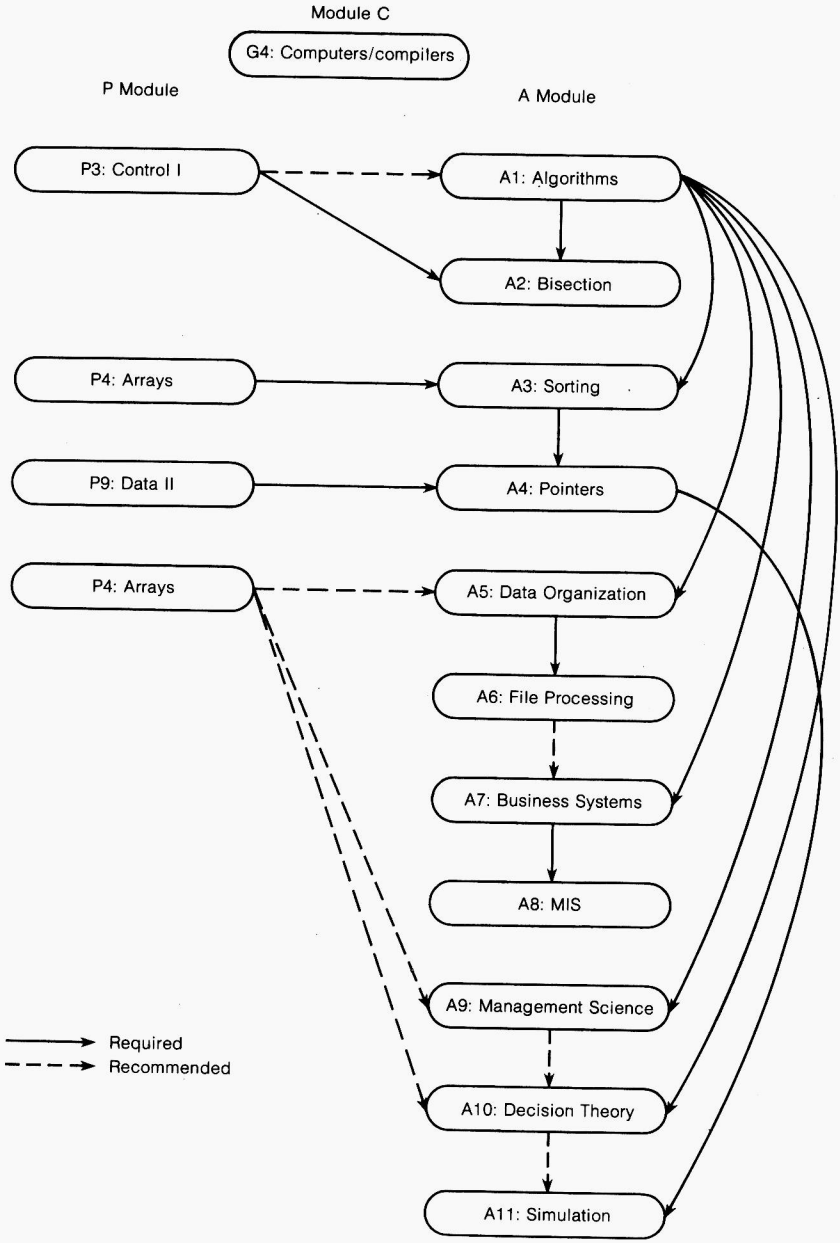
Preface

This version of Module A (Algorithms and Applications) is intended for use by business students. Using the programming principles and informal language introduced in Module P (Programming and Languages), a variety of techniques and methods of solution are developed that are useful in broad classes of numerical and nonnumerical problems. The material is organized so that later chapters depend minimally on earlier ones, to allow the instructor flexibility in the selection and ordering of topics. To help the instructor, a diagram appears at the beginning of this module, showing specific prerequisites for each chapter.

For a one-semester course, Module A can be used to amplify the material in Modules P and C by selecting example applications of interest to the students. Alternatively, Modules A, C, and P together provide enough material for a two-semester sequence in business data processing, computer programming, and management science. In the first semester, Chapters A1 and Chapters A5 through A10 of this module can be covered, along with most of Module C and much of Module P, while a language is being taught. The remainder of this module and Module P can be covered in the second semester, along with a second language if desired. The language manuals are so closely tied to Module P that few additional ideas would have to be introduced: essentially it would be necessary only to provide exercises in the syntax of the second language.

This module can also be used in a separate course in business data processing if the students have an adequate background in programming in a nontrivial programming language. The informal language used to describe algorithms is natural for anyone with moderate exposure to a structured language; for students with experience only in Fortran or Basic, a quick review of the General Introduction and a brief discussion of the basic structured language constructs would be needed.

This module teaches not only business applications but also the concepts upon which the applications are based. Accordingly, this module is a useful supplement for courses in computer science, mathematics, and engineering students. Many of the business data processing techniques in Chapters A5 through A8 and the management science techniques in Chapters A9 through A11 depend heavily on mathematics and other disciplines.



Prerequisite structure for Module A

Contents

Module

A

Algorithms and Applications in Business

The purpose of a computer is to perform calculations that yield answers to particular cases of general problems. These problems arise in many application areas. Business applications include *data processing*, the manipulation of large amounts of information—for example, to update computer-stored files and generate reports and records of individual business transactions (weekly pay slips, orders, invoices, airline tickets, and so on). Another application is *information retrieval*, which allows access to data in computer-stored files, to show potential trouble areas quickly. A system that offers this function is called a *management information system* or MIS. Business applications also include operations-research techniques such as simulation—the numerical representation or modeling of a business problem. For example, a company may decide where to put a new piece of machinery on the basis of a computer simulation of various new plant layouts. The simulation enables the company to find the layout that will yield the maximum return on the investment.

Scientific and engineering applications include the approximate solution of problems using numerical techniques. This is really a kind of simulation, because the mathematical equations that are solved are models of the real world; they represent the way the engineer or scientist believes a process behaves. The engineer and scientist are also interested in data processing if they are confronted with large amounts of experimental data that must be correlated and compared. They can make good use of information retrieval capabilities if they are working with large amounts of textual data.

As you can see, different disciplines have similar problems that can be solved by similar methods. In this module, we examine problems such as sorting, searching, and pointer manipulation that are common

to many application areas and develop methods of solution. In addition, we examine some specialized business applications.

Chapters A1 and A2 concentrate on algorithmic techniques that are used for all data structures. The illustrations in these chapters involve only simple variables. Chapters A3, A4 and A5 introduce fundamental algorithms that operate on more complex data structures. Chapter A1 includes samples of business applications as examples of general types of algorithms, and discusses general types of algorithms. Chapters A5 through A8 introduce data processing and information systems. Chapters A9 through A11 introduce operations research/management science techniques that aid in business decision making.

Chapter

A1

Types of Algorithm

There is no universal set of rules for designing algorithms: each new problem may need a totally new approach. Indeed, it is this aspect of computer programming that can be the most pleasurable, providing a challenge akin to a crossword puzzle or chess problem and giving an outlet to the ingenuity and creativity of the programmer. There are, however, a number of basic *types* of algorithm that can frequently be used to solve a particular problem.

Five common types of algorithm are given below, followed by examples and a discussion of each:

- *Direct computation*—in which the exact answer is obtained by a sequence of elementary computations.
- *Trial and error*—a type of iteration in which each successive approximation is based on the degree of error in the previous approximation.
- *Divide and conquer*—in which the problem is divided into similar but smaller problems that can either be solved directly or be further subdivided by the same technique.
- *Enumeration*—in which all possible "answers" are tried in order to find one that solves the problem.
- *Iteration*—in which a series of increasingly precise approximate answers are computed until one is obtained that is "close enough." (An exact solution would require an infinite number of operations.)

A1.1 DISCUSSION AND EXAMPLES OF ALGORITHMS

Direct computation. The income-tax computation of Chapter G2 is an example of direct computation. This form of solution is applicable to simple problems in which the problem description itself specifies the computation needed to solve the problem.

Example A1.1 *Calculating Profit*

Calculating the total profit that results from selling an item is an example of a direct computation algorithm. For instance, a bookstore may want to calculate the total profit resulting from the sales of one book in its inventory. Total profit (TOTPROF) is calculated by subtracting the total cost (TOTCOST) of an item from the total revenue (TOTREV) earned by the item,

TOTPROF←TOTREV−TOTCOST

To find the total profit, we first need to know the total revenue and total cost. These are calculated by multiplying the price per item (PRICE) and the cost to the store per item (COST) by the number of units sold (UNIT). In other words,

TOTREV←PRICE*UNIT
TOTCOST←COST*UNIT

To calculate TOTPROF, we need values for PRICE, COST, and UNIT. From these, we can calculate TOTREV, TOTCOST, and TOTPROF. An algorithm for these calculations is shown in the structured flowchart in Figure A1.1.

Let's return to the bookstore example. Suppose 27 copies of a book that costs \$6.50 are sold for \$9.25. The input would be

27, 6.50, 9.25

and the algorithm would compute the following values:

$$\begin{aligned} \text{TOTREV} &= 9.25 \times 27 = 249.75 \\ \text{TOTCOST} &= 6.50 \times 27 = 175.50 \\ \text{TOTPROF} &= 249.75 - 175.50 = 74.25 \end{aligned}$$

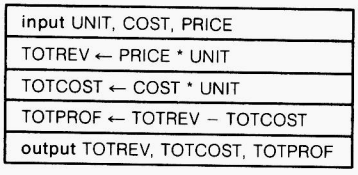


Figure A1.1 Structured flowchart of program to calculate profit

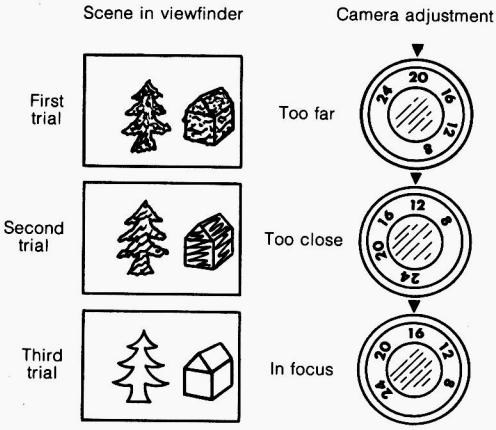


Figure A1.2 Trial-and-error adjustments in focusing a camera

Trial and Error. In trial-and-error algorithms, the amount by which a current approximation fails to satisfy the problem is used to determine the next approximation. This process is similar to the way people perform many everyday actions—driving a car, focusing a camera (see Figure A1.2), or almost any action that involves movement.

While driving a car, for example, the driver steers by turning the wheel, observing whether more or less turn is needed, and adjusting the wheel accordingly. The driver first makes a trial attempt, and then corrects to reduce the error. If forced to drive blindfolded, the driver would not be able to steer correctly, because the measurement of the error is essential to the correction. Chapter A2 gives a trial-and-error method for solving problems common to many business and scientific applications.

Divide and conquer. Breaking a problem into simpler subproblems is a very powerful technique, useful for both numerical and nonnumerical problems. We will illustrate it with a search in an ordered list, such as a telephone book. One way of doing such a search is to open the book in the middle and see whether the item sought is before or after the middle entry. This can be done by a single comparison with the middle entry, because it is known that all items before that entry are alphabetically less and all items after it are alphabetically greater. Thus in one step we have reduced the size of the list to be searched by half. The same technique can now be applied to the smaller list. Thus, if the original list had 16 items in it, the first comparison leaves us with a list of 8 items to consider, the second with a list of 4, the third with a list of 2, and the last with a list of 1. A list of one item can be searched very quickly indeed! This particular search method, called a *binary search*,

is a very important technique and is the basis for many related algorithms. We will be discussing it in more detail in Chapter A3.

Enumeration. A sequential search is an example of enumeration: each entry in a list is checked to see if it is the one sought. Enumeration is usually very slow, but sometimes it is the only method available. Often it is possible to start with an enumeration method and improve it by avoiding obviously impossible cases, as the following example shows.

Example A1.2 ***Prime Numbers***

Given a positive number N greater than 1, find the smallest integer $M > 1$ that divides N exactly.

If the smallest divisor is N, then N must be prime. An enumerative method for solving this problem is simply to test each integer less than N, starting with 2, to see if it divides N. If one is found, it is the smallest. To program this solution, we need to be able to test whether M divides N. This is a basic operation in some computers and programming languages, but not in others. However, it can be programmed in terms of more elementary operations by testing whether N is equal to (N ÷ M) * M (see Chapter P2). Our first attempt at this program is shown in Program A1.1. If N is prime, the loop is executed $N - 2$ times (for the values $M = 2, 3, \ldots, N - 1$). A little thought reveals that if N is not prime, one of its divisors must be less than or equal to the square root of N, so there is no need to test any values above that. Program A1.1a gives a revised version. For the case $N = 127$, Program A1.1 executes its loop 125 times, whereas Program A1.1a executes its loop only 10 times. A further improvement is possible by checking only for $M = 2$ and the odd numbers between 3 and the square root of N.

Enumeration methods are the basis of many programs for nonnumerical problems, but because they can be so slow, it is essential to conduct a careful analysis to look for improvements.

Program A1.1 ***Find a divisor of*** N

```
SMALLEST_DIVISOR: program
    integer M,N
        M←2
        do while M*(N÷M)≠N
            M←M+1
            enddo
        output M
    endprogram SMALLEST_DIVISOR
```

Program A1.1a ***Improved divisor program***

```
SMALLEST__DIVISOR: program
    integer M,N
        M←2
        do while M↑2≤N and M*(N÷M)≠N
            M←M+1
            enddo
        if M↑2>N then M←N endif
        output M
    endprogram SMALLEST__DIVISOR
```

Iteration. Iteration techniques are usually applicable to numerical problems. An example is the computation of a function such as sin(X). It can be shown that the value of sin(X) is given by the expression

$$\sin(X) = X - X^3/3! + X^5/5! - X^7/7! + \ldots$$

(where 5! means $5 \times 4 \times 3 \times 2 \times 1$, or *factorial* 5). This does not lead to a direct algorithm, because it requires an infinite number of operations. However, for any desired degree of precision, it is sufficient to use only the first part of the infinite sequence. In particular, if we are content with a precision of $\pm 10^{-5}$ for all values of X between -1.0 and $+1.0$, it can be shown that we can use

$$\sin(X) \cong X - X^3/3! + X^5/5! - X^7/7!$$

This computation requires only a finite number of operations, and can now be coded directly. If more precision is needed, additional terms can be added. For example, the next term ($X^9/9!$) should be added if an accuracy of 10^{-7} is required for the same values of X. It can be shown for this example that the desired precision can be achieved by including all terms until a term is generated that is smaller than the error allowed, so a program can be written to *iterate* until the desired accuracy is obtained, as shown in Program A1.2.

Program A1.2 ***Compute sine by iteration***

```
SINE: program
    The sine of a number X is computed using a power series. Terms
    are added until the next term is less than ERROR.
    real SINE,X,ERROR,NEXT__TERM,I
        SINE←X
        I←4.0
        NEXT__TERM←−X↑3 / 6.0
```

```
        do while ABS(NEXT__TERM)≥ERROR
            SINE←SINE+NEXT__TERM
            NEXT__TERM←−NEXT__TERM*X↑2 / (I*(I+1.0))
            I←I+2.0
            enddo
        output 'SINE OF',X,'IS',SINE
    endprogram SINE
```

In Program A1.2, the variable SINE is used to accumulate the value of the approximation to sin(X). Alternate negative and positive values, which are always decreasing in absolute value, are added to SINE in successive passes through the loop. The program takes advantage of the fact that, in the infinite series representation of sin(X), each term to be added can be obtained from the previous term. For example,

$$\frac{-X^7}{7!} = \left(\frac{X^5}{5!}\right) \times \left(\frac{-X^2}{7 \times 6}\right) \text{ uses } \frac{X^5}{5!} \text{ to compute } \frac{-X^7}{7!}$$

and

$$\frac{X^9}{9!} = \left(\frac{-X^7}{7!}\right) \times \left(\frac{-X^2}{9 \times 8}\right) \text{ uses } \frac{-X^7}{7!} \text{ to compute } \frac{X^9}{9!}$$

A1.2 DEVELOPING ALGORITHMS FOR BUSINESS

Now that we've discussed the different types of algorithms, let's examine how they can be used in business situations. Algorithms are developed to solve specific problems. The first step in developing an algorithm is to understand the problem to be solved and its method of solution; this often is the most difficult part of writing a program.

As an example of algorithm development, consider how an accountant's procedures for determining depreciation can be transformed into computer programs. The physical assets of a business, such as machinery, buildings, and equipment, tend to wear out, and must be discarded; they depreciate in value. Businesses must account for the depreciation of physical assets as they determine the cost of doing business.

There are several means of determining depreciation. They all follow a similar pattern. The difference between the original cost of an asset (its *book value*) and however much the asset can be sold for at the end of its useful life (its *salvage value*) is apportioned over the life of the asset. Depreciation methods vary in how much loss in an asset's value they attribute to any given period of time. Some methods divide depreciation equally across the life of the asset; others provide for faster depreciation in the early life of the asset, and slower depreciation later.