Introduction to Computer Science

A Structured Approach / Neill Graham

Neill Graham

Introduction to Computer Science

A Structured Approach

WEST PUBLISHING CO.
St. Paul New York
Los Angeles San Francisco

50 West Kellogg Boulevard
P.O. Box 3526
St. Paul, Minnesota 55165

Printed in the United States of America

Library of Congress Cataloging in Publication Data

Graham, Neill, 1941-
Introduction to computer science.

Bibliography: p.
Includes index.
1. Structured programming. 2. FORTRAN (Computer program language) I. Title.
QA76.6.G68 001.6'42 78-11196
ISBN 0-8299-0187-6

3rd Reprint—1980

Preface

This book is a text for an introductory computer science course for first- or second-year college students. It can be used for either a one-semester or a two-semester course. There is more than enough material for the one-semester course, giving the instructor the freedom to choose those topics most suitable for a particular class. If supplemented by additional material on FORTRAN or some other programming language, the book contains sufficient material for a two-semester course as well.

Most of this book requires no mathematical preparation beyond the usual elementary and high school courses. No previous acquaintance with variables or expressions is assumed, so the student who has forgotten, or never had, high school algebra should not be at a disadvantage.

The last two chapters of the book are on numerical methods. They necessarily assume somewhat more mathematical preparation, about equivalent to the introductory college courses in algebra and trigonometry. Although some ideas from calculus are introduced, no previous knowledge of the subject is assumed.

Many current computer science texts use flowcharts as the principal means of presenting algorithms. This book, following the recommendations of the advocates of structured programming, uses an informal *algorithmic language* or *pseudocode* instead. In the chapters on algorithm construction, however, flowcharts are also used, both to introduce the student to them and to help illustrate the basic control structures. Later, the flowcharts are dropped, and the student is led to rely on the algorithmic language alone.

The algorithmic language is intended as an *informal notation* rather than a rigidly specified programming language. Although beginners will do well to use the language as it is described in the text, the instructor and the advanced student should not hesitate to modify features or add new ones as might be required by a particular problem.

Data-type declarations are optional in the algorithmic language and are usually omitted. Data-type information can be easily presented in informal comments, either in the algorithm or in the surrounding text. There seems little need to burden the student with additional formal machinery for this purpose.

Algorithms are printed using uppercase letters only. Students who have previously encountered a programming language such as BASIC or FORTRAN will find this format more familiar (and perhaps less threatening) than the Algol-like boldface-italics format. There is certainly no need for students to print their own algorithms in uppercase, however; ordinary handwriting may be used as suggested in Fig. 4-1. Key words may be underlined if desired.

Supplements are available showing how to translate from the algorithmic language into common programming languages. The FORTRAN supplement is included in the book as an appendix; the other supplements are available separately.

The book begins with an introductory chapter which presents an overview of computers, information processing, algorithms, programs, and flowcharts.

Part 1 surveys computer hardware and software. The software chapter goes somewhat more deeply into the subject than is usual in an introductory course. The unifying idea is that of an abstract machine, which can be implemented in hardware (a physical machine) or software (a virtual machine).

Part 1 may be omitted or delayed by instructors who prefer to get their students writing programs as soon as possible.

Part 2 on algorithm construction is the heart of the book. Variables, values, and expressions are introduced, as are the basic control structures of sequence, selection, and repetition. A chapter on subroutines and functions is included, as well as one on algorithm design and testing.

Part 3 shifts the emphasis from control structures to data structures. Arrays were introduced in Part 2 to demonstrate additional applications of repetition. Now strings, stacks, records, linked lists, trees, and graphs are taken up. The chapter on trees further develops the ideas of recursive programming introduced in Part 2.

Part 4, on files, is an introduction to the ideas of data processing and information retrieval. For sequential files, the classical file update problem is discussed, as are several methods of external sorting. For random files, the emphasis is on retrieving designated records. Methods discussed include hashing and indexed sequential access methods for primary key retrieval as well as multilists and inverted files for secondary key retrieval.

Part 5, on numerical methods, takes up the half-interval method for

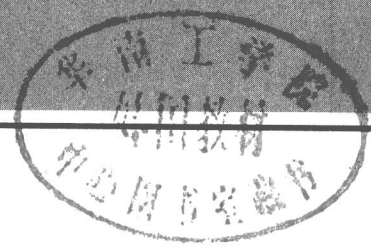
solving nonlinear equations as well as the Gauss-Jordan and Gauss-Seidel methods for systems of linear equations. Graphs are used to illustrate the convergence problems of the Gauss-Seidel method.

The final chapter of the book is devoted to numerical integration. To make this material accessible to the student who has not studied calculus, the notation and terminology of calculus are avoided. Instead, we concentrate on the physically intuitive idea of a moving object whose velocity is specified as a function of position and time, or whose acceleration is specified as a function of position, velocity, and time. While calculus would allow some results to be stated with greater precision, it would also deny the noncalculus student insight into an important computer application.

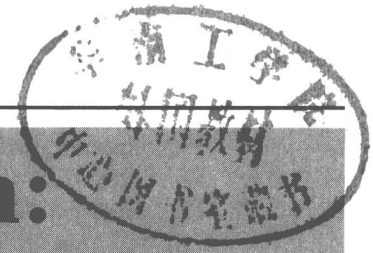
I would like to thank the following persons, who read various drafts of the manuscript, for their valuable comments, criticisms, and suggestions: Edward Bowdon, Randy Byers, Nell Dale, Robert Dourson, Paul Emerick, Olin Johnson, William Moldrup, Rex Page, and Michael Stimson.

*

Contents



†



1 Introduction: Computers, Information, and Algorithms

Exactly what is a computer? What can it do? Are computers really just superfast adding machines, or can they do other things besides arithmetic? What is a program, and what do computer programmers do? What kinds of data can a computer process?

Although computers differ vastly in their size, cost, internal construction, and external appearance, they all have two characteristics in common:

- A computer is an information-processing machine. It manipulates information just as some other machines manipulate wood or plastic or steel.
- In carrying out an information-processing task, the computer is controlled by a set of detailed, step-by-step instructions called a *program.*

In this chapter we will examine each of these two characteristics of computers in detail.

1.1 THE INFORMATION PROCESSING MACHINE

Information. Let us begin with the nature of information itself. We can best think of information as "knowledge in motion." That is, information is knowledge in the process of being carried from one person to another. If one person gives information to another, then the receiver's knowledge is increased by the information received. Happily, the reverse is not true: The sender does not lose any knowledge by giving information to someone else.

Not all information originates with people. A scientist, for instance, extracts information directly from nature through observations and measurements. But this is the exception rather than the rule. Most information passes *ultimately* from person to person.

But a variety of inanimate objects can intervene between sender and receiver. Books, motion pictures, radio, television, telephones, and computers are familiar examples of this. Most of these devices merely attempt to transmit or to store the original information as faithfully as possible. Computers are unique because they often modify the information as well as store or transmit it.

When we say that receiving information *increases* a person's knowledge, we are forgetting about *misinformation.* Receiving untrue information will *decrease* your knowledge. That is, it will worsen the correspondence between what is inside your head and the facts of the outside world. We can only say that receiving information changes a person's knowledge. Whether the change is an increase or a decrease depends on the particular information received.

A good-for-examinations definition of information, then, is anything which produces a change in a person's knowledge.

Symbols. So far, our discussion has been terribly abstract. We have yet to come up with anything we could actually point to and say, "Over on your left, ladies and gentlemen, in the glass case, is one of the finest specimens of knowledge it has ever been my privilege to see. And if you will move this way, please, I will point out our most recently acquired piece of information."

And yet we do not use abstractions to communicate with one another! We use facial expressions, bodily movements, inarticulate grunts, groans, cries, and laughter, spoken English (and other languages), notes, memos, letters, telegrams, telephone calls, pictures, and many other concrete, physical things.

In short, before information can be passed from one person to another, it must be represented in concrete, physical form. The physical things we use to represent information are called *symbols.* The letters of the alphabet, the sounds we make in speaking, the electrical currents that travel over telephone wires, and the electromagnetic waves our TV sets pick up are all examples of symbols. Figure 1-1 gives some further examples.

Anything that we do with information must actually be done with the concrete symbols that represent the information. To write a love letter, for instance, you must make a large number of marks on a piece of paper with pen or pencil. Whatever the emotional content of the letter, that content can only be expressed by arranging marks on paper in the appropriate order.

Data. In computing, we refer to a series of symbols as *data.* Data is normally used to represent information, and the terms *data* and *information* are often used rather interchangeably. But notice that the data

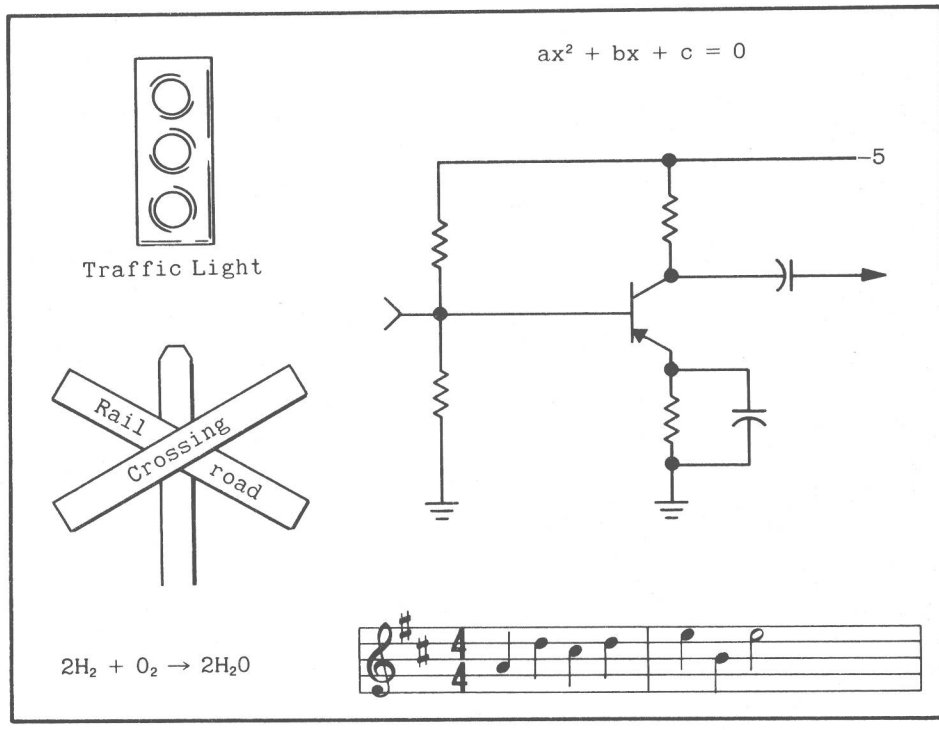


Fig. 1-1 Some commonly used symbols.

```
I LOVE YOU
```

represents information, while the data

```
XQW RTYU XDHFDI
```

does not, at least if we assume that valid information is to be expressed in English. Such meaningless data is affectionately known as *garbage*, and no one can work around computers very long without encountering it in substantial quantities.

By the way, the word *data*, by derivation, should be plural, the word *datum* being the singular form. But computer people use *data* as a mass noun, like *grass* or *sand*. We say "the data has been processed" just as we say "the grass has been cut" or "the sand has been shoveled."

Symbol Manipulation. We said that a computer was an information-processing machine. But the computer cannot process information in the abstract; it can only manipulate the concrete symbols which represent that information. We might better describe a computer, then, as a *symbol-manipulating machine*. Information processing is the abstract idea we have in mind. Symbol manipulation is the means by which that idea can be realized in practice.

A computer can process any kinds of symbols that can be translated into electrical signals. That includes most kinds, these days. In addition to the letters, digits, and punctuation marks of the alphabet, computers can process pictures, drawings, speech, music, and electrical sensing and control signals from other machines.

A computer will manipulate meaningless symbols just as cheerfully as meaningful ones. "Garbage in; garbage out" is an old saying among computer people, and one that we have frequent occasion to use.

Examples of Information Processing. The kinds of information processing tasks a computer can carry out are so varied that it is almost impossible to summarize them in words. But the following examples will give you some idea of the computer's versatility.

- *Arithmetical calculations.* The digits 0 through 9 are symbols, and the familiar operations of arithmetic are manipulations that can be carried out on those symbols. Until recently, arithmetic was the only form of symbol manipulation that was to any degree mechanized.

Computers can indeed do arithmetic very fast and very accurately. This has consequences of which most people are unaware. For instance, one reason you can now purchase relatively inexpensive cameras with excellent-quality lenses is that those lenses were designed with the aid of a computer. For each trial design, the computer calculated the paths of thousands of light rays through the lens. If the calculated paths were unsatisfactory, then the computer made changes in the trial design and calculated a new set of paths. The process continued until a satisfactory design was found, or until the computer gave up because a predetermined number of trials had yielded no satisfactory solution.

Perhaps because "compute" originally meant "to do arithmetic," people seem to think first of arithmetic when they think of computers. Computers were invented to solve mathematical problems, and this remains one of their important applications. But they can do other jobs, too. Many computer applications involve little or no arithmetic.

- *Word Processing.* You type the first draft of a letter, a term paper, an article, or a book into the computer. You then revise by directing the computer to insert, delete, and rearrange specific pieces of material. When the result meets your expectations, you can order the computer to type out a perfect copy.

- *Game Playing.* A computer, particularly one equipped with a television-like display terminal, makes a fantastic "gameboard." Computers are widely used for game playing by computer hobbyists—people who own their own personal computers. One of the most common computer games is a space war game based on the television series *Star Trek.*

Besides just serving as a gameboard, the computer can take an active part in the game, playing against one or more human opponents. One computer program, for instance, plays checkers at the championship level. Another that plays chess has been improved over the years from the class C to the class B level, and now sometimes defeats expert players.

- *Controlling Machines.* Computers can be used to operate other machines. For example, some late-model cars use a computer to "tune"

the engine continually while the car is running. The computer monitors such things as engine speed, power demand, and power output. It adjusts fuel-feed rate, fuel-air mixture, and spark timing for best performance. Not only does this constant tuning improve performance, it reduces pollution to the point where such antipollution devices as catalytic converters can, in many cases, be eliminated.

- *Computer-Assisted Instruction.* The computer presents a student with a small segment of a lesson, and one or more questions on that segment. If the student answers correctly, the computer will move on to the next segment. Otherwise it will present additional review material based on the incorrect answers. Thus, a student is "led by the hand" through a subject, exposed only to the material that he needs to achieve understanding. While this is not as satisfactory as a personal tutor, it may be the next best thing.
- *Data File Management and Information Retrieval.* A computer can maintain large data files and retrieve individual items from those files upon request. A researcher, for instance, can enter into the computer a list of key words describing his interests; the computer will return a list of all papers in the researcher's field which contain one of the key words in their titles or abstracts. Or a doctor can type in a list of symptoms, and get back the latest information on diseases having those symptoms.

1.2 THE INSTRUCTION-FOLLOWING MACHINE

An All-Purpose Machine. A computer carries out a particular information-processing task by following a set of detailed instructions. To change the computer's task, all we have to do is change the instructions.

The catch to this is that the computer will not perform *any* task until it is provided with the necessary instructions. This does not mean, however, that the person who uses the computer will have to supply the instructions. Instructions for doing many tasks may be available from the computer manufacturer or from other sources. Some computers even come with the instructions for doing a particular task permanently installed, so the computer will always do that task and no other.

People sometimes say, "a computer can only do what it is told to," as if this were some limitation on the machine. Quite the contrary! To be sure, a dishwasher does not have to be told how to wash dishes, nor a lawnmower how to mow lawns. But then a dishwasher can *only* wash dishes and a lawnmower can *only* mow lawns. A single computer, on the other hand, can compute the orbit of a spacecraft, play a game of chess, or make out your paycheck, provided only that it is given the proper instructions.

One of the founders of computer science, the Hungarian-American mathematician John von Neumann, said that the computer is the "all-purpose machine." We might qualify this to "all-purpose information-processing machine" (after all, it won't mow the lawn). That the computer can "do what it is told to" is precisely what makes it "all-purpose."

Algorithms. We now turn our attention to the instructions that a computer needs to perform its tasks. A set of instructions for accomplishing a particular task is called an *algorithm.* Synonymous terms are *method, technique, procedure,* and so on. An algorithm that is intended for execution by a computer is also called a *program.*

Characteristics of Algorithms. All algorithms share the following three features:

- *An algorithm is precise.* Each step of an algorithm must specify exactly what action is to be carried out. There is no room for vagueness. Also, every step must be stated explicitly. None can be "understood" or "assumed."

Since it is difficult to achieve the necessary precision in English, a number of *algorithmic languages,* including *programming languages,* have been devised. These are analogous to the notations used in music, mathematics, dance, chemistry, knitting, and crocheting to express technical ideas more concisely and precisely than is possible in English.

- *An algorithm is effective.* It must be possible for the person or machine executing an algorithm to carry out its instructions effectively. Suppose, for instance, that an algorithm demanded that we take the square root of 2 with perfect precision. The square root of 2 is given by

$$\sqrt{2} = 1.4142135623\ldots$$

where the dots indicate an infinite sequence of additional digits. This unending sequence of digits could never be worked out in a finite amount of time, could never all be written out on a blackboard or a piece of paper, could never all be stored inside any computer. An algorithm that demands we take the square root of 2 with perfect precision, then, is not effective.

- *An algorithm must terminate.* When a person or machine executes an algorithm, he, she, or it must eventually reach the point where the task is complete and no more instructions remain to be executed. The execution of an algorithm must not go on indefinitely.

You may find it hard to see how an algorithm with only a finite number of instructions could ever fail to terminate. Surely after we have executed each instruction, there will be nothing else left to do, and the execution will have to terminate.

The catch is that an algorithm may demand that some instruction be carried out repeatedly, until a given condition occurs. "Stir the sauce gently over low heat *until* it comes to a boil," for instance. If the terminating condition never occurs, the repetition will continue indefinitely. Suppose we had merely said, "stir the sauce gently until it comes to a boil," without saying that it should be heated. Someone who took the algorithm literally—and computers *always* take algorithms literally—would have a lot of stirring to do.

The Euclidean Algorithm. As an example, let us take one of the oldest recorded algorithms, Euclid's algorithm for finding the greatest common divisor of two numbers.