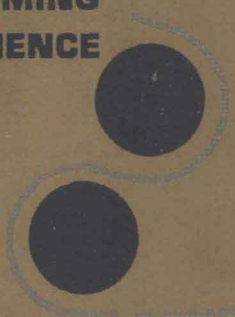
INTRODUCTION TO PROGRAMMING AND COMPUTER SCIENCE



INTRODUCTION TO PROGRAMMING AND COMPUTER SCIENCE

Anthony Ralston
Chairman, Department of Computer Science
State University of New York at Buffalo

McGraw-Hill Book Company
New York, St. Louis, San Francisco, Düsseldorf, Johnannesburg,
Kuala Lumpur, London, Mexico, Montreal, New Delhi,
Panama, Rio de Janeiro, Singapore, Sydney, Toronto

INTRODUCTION TO PROGRAMMING AND COMPUTER SCIENCE

Library of Congress Catalog Card Number 78-136182
07-051161-6
3 4 5 6 7 8 9 0 M A M M 7 9 8 7 6 5 4 3 2

This book was set in Journal Roman by Creative Book Services, division of McGregor & Werner, Incorporated, and printed on permanent paper and bound by The Maple Press Company. The designer was Creative Book Services. The editor was Richard F. Dojny. Loretta Palma supervised production.

INTRODUCTION
TO PROGRAMMING
AND COMPUTER SCIENCE

McGRAW-HILL COMPUTER SCIENCE SERIES

RICHARD W. HAMMING
Bell Telephone Laboratories

EDWARD A. FEIGENBAUM
Stanford University

Bell and Newell *Computer Structures: Readings and Examples*
Cole *Introduction to Computing*
Gear *Computer Organization and Programming*
Givone *Introduction to Switching Circuit Theory*
Hamming *Introduction to Applied Numerical Analysis*
Hellerman *Digital Computer System Principles*
Kohavi *Switching and Finite Automata Theory*
Liu *Introduction to Combinatorial Mathematics*
Nilsson *Artificial Intelligence*
Ralston *Introduction to Programming and Computer Science*
Rosen *Programming Systems and Languages*
Salton *Automatic Information Organization and Retrieval*
Watson *Timesharing System Design Concepts*
Wegner *Programming Languages, Information Structures, and Machine Organization*

To ELIZABETH

May she grow up to
read this with some
of the pleasure with
which I dedicate it

preface

You must talk to the media, not to the programmer. To talk to the programmer is like complaining to a hot dog vendor at a ballpark about how badly your favorite team is playing.

Marshall McLuhan

This book is concerned with talking to computers. It has been used by me and some of my colleagues at other universities in recent years as a text for a one-semester first course in computing. One of my basic premises in teaching such a course, which I hope is reflected in this book, is that it is time—past time, really—for the first course in computer science to be taught at an intellectual level similar to that of first university courses in other disciplines. This creates some special problems in computer science. Not only will a first course be taken by undergraduates majoring in an increasingly wide spectrum of disciplines, but, for the foreseeable future, also by undergraduates at all levels from, say, freshmen in the sciences to seniors in the arts and humanities. This book is not aimed specifically at any segment or level of the undergraduate population: it is my position that much the same first course in computing should be given to all undergraduate students, except, perhaps, for an honors-type course for concentrators in computer science. This is not to say that precisely the same subject matter is best for both the mathematics major and the English major, but that the differences are not so great that they cannot be accommodated by one book or in a single class. It is also not to say that courses in the social implications of computers are not good ones for many liberal arts majors, at least. They can be. But such courses are perhaps made better if the students have previously learned enough about computers qua computers to grasp fully these implications.

Almost all colleges and universities now offer an introductory course in computing or computer science. This course is taught from many points of view and with varying aims, which is as it should be at this stage of computer science education. Among these aims the following are easily distinguishable:

1. To teach only a particular computer language.
2. To teach the student what a computer algorithm is, while teaching some computer language to be used as the tool by which to convey the algorithm to the computer.
3. To introduce the student to the concept of communication with a computer by emphasizing the comparison between structures designed to do similar tasks in various languages, while again emphasizing the teaching of one language in depth and getting the idea of an algorithm across intuitively as a by-product of using the computer language or languages.

The first approach is only too reminiscent of the courses in the use of the slide rule which were formerly given to engineers. Although a student in a first course in computing should *at least* learn to use a computer via some language, a course which teaches no more than this is almost devoid of intellectual value and it is doubtful if it is worthy of college credit.

The latter two approaches both have their proponents and certainly both are defensible. Courses embodying either approach typically include an introduction to a variety of other topics in computer science. My own feeling is that, except perhaps for those who will become professional computer scientists, an ability to communicate and an understanding of the mechanism of communication should be the aim of this first course. Most students in the course will never be more than users, however sophisticated, for whom formal notions of an algorithm are mostly irrelevant since constant use will give them a sound intuitive notion of an algorithm.

This book, therefore, concentrates on computer languages, their major components, and how these components are implemented in some languages. No attempt is made to teach any specific language. I assume that any course using this book as a text will, in addition, use my *Fortran IV Programming—A Concise Exposition,* if Fortran is the language the students will actually use, or one of the many good books on the other three languages discussed here (Algol, PL/I, and Cobol). I should note my belief that whatever supplementary book is used, it should be used much more as a reference than directly as a text to lecture from. I firmly believe that the purpose of the lectures for this course (or, indeed, for any course) should not be to dot all the *i*'s or cross all the *t*'s, but rather to explain subleties and interrelations. One result of any college education should be an ability to study and learn the prosaic details of a subject on one's own; much of the study of any computer language falls in this domain.

In one sense this book is a text in comparative linguistics. Of the four languages considered, one, Cobol, is treated quite cursorily because my orientation is certainly more toward scientific data processing than business data processing. In addition, despite the very large current investment in Cobol programs in the United States, it seems likely that the basic cumbersomeness and limitations of this language will gradually cause it to be replaced by another language, perhaps PL/I. Aspects of Algol and PL/I are treated in substantially more detail than for Cobol but in illustrations and examples Fortran is used somewhat more than Algol or PL/I. This reflects not only the fact that it is the language I know best but also the fact that Fortran is now and, for the near future at least, will continue to be the most widely used language at American universities.

The four languages discussed in this book are all examples of higher level languages and, more specifically, of procedure-oriented languages. Because there are still proponents of the point of view that lower level languages, that is, machine and assembly languages, should be taught before higher level languages, it is appropriate to say a few words on this matter here. Indeed, in the early days of digital computers, it made a lot of sense to teach machine language to all users of computers just as it behooved all drivers of the first automobiles to know something about internal combustion engines. But, just as most of us now learn to drive without knowing very much about automobile engines, neither is there any reason to learn machine or assembly language—which implies learning some specifics about the computer—before learning a higher level language. In fact, in order to "drive" (i.e., to use effectively) a computer, the vast majority of all users will never need to know more than a higher level language. Of course, as it is useful to a driver to know how to check the water level in the radiator, so is it useful to the higher level language computer user to know certain basic facts about the structure and organization of computers; these are discussed in this book. Machine and assembly language is best left to a second or later course in computer science.

Given that a student should begin with higher level languages, why not be content with considering only one language? After all, most of us learn to talk only a single natural language initially. The most obvious answer to this is, since we learn our natural language intuitively as regards speaking, reading, and grammar, and then use these intuitive (and partly taught) concepts when we study a foreign language, there is just no valid analogy between learning natural and computer languages. From my point of view a better answer is obtained by analogy with children who grow up bilingually. Such children seem to go through a stage of confusion during which they cannot distinguish in their own speech between the two languages but, at a fairly early age, this confusion disappears and their language skills are usually markedly superior to monolingual

children. I have no doubts, both from intuition and experience, that teaching students about more than one language simultaneously introduces an element of confusion and makes the course more difficult. But I do believe the result is a better, deeper understanding of computer languages and their structure, the most important by-product of which is an increased ability to learn other computer languages.

In addition to the computer-language aspects of this book, there is a liberal sprinkling of other topics which I believe will give students a broader view of what computer science is really about. Among these are discussions of computer arithmetic, compiling of arithmetic expressions, and logic, which I would expect the individual instructor to take or leave depending upon his own predilections. I do feel strongly that discussion of these or related topics in a first course on computers adds substantially to the intellectual value of the course and gives the beginning student a better basis on which to accept or reject computer science as a possible career.

One of the most difficult problems I had in writing this book was to try and produce something which could reasonably be followed in a linear fashion in a course. The conflict is a familiar one to all instructors of beginning computer courses. On the one hand, for motivational reasons, there is much to be said for getting the student on the computer (i.e., giving him some kind of problem to solve using the computer) as early as possible and to have him continue to use the computer throughout the course. This usually involves introducing certain concepts early in the course in a cursory or oversimplified fashion and then only really explaining them later on. On the other hand, writing a book does not lend itself to Ping-Ponging back and forth between topics. Some order in a written presentation is, I believe, necessary and desirable, and this involves a fair amount of introductory material before getting to those topics directly involved with using the computer. This can—and has, in some recent books—lead to devoting the first half of the book to nonlanguage topics.

I have partly solved this problem by restricting the introductory material considerably, omitting, in particular, almost all discussion of computer applications on the assumption that students are more and more aware of these and that, anyhow, individual instructors can handle this better than it can be done in print. But there is still enough introductory material so that it takes two or three weeks to get into the computer-language aspects themselves. In order not to have to wait a few weeks to get the students on a computer, at the first meeting of the course I give each student a deck of cards consisting of a source program and necessary data for a simple calculation (e.g., solution of quadratic equations, a compound interest calculation). The source program is written to be as nearly self-explanatory as possible to someone with no computer background

whatsoever. Each source deck contains some syntactic or logical error such that the error message or incorrect output should enable the complete novice to correct the program. In the first lecture the students are taught which control cards are necessary to complete the deck; later they keypunch these cards, run the program, and rerun it until the error is corrected.

But it is still not possible to give the students their first *real* computer project as early as I would like (no later than the beginning of the third week) and still follow the text as written. My solution to this problem is embodied in the aforementioned *Fortran IV Programming—A Concise Exposition,* which begins with a section covering enough basic aspects of the Fortran language to enable students to be given meaningful computer projects early in the semester. I use this section for a few lectures in the second and third week of the semester and then return to this book. When the appropriate part of this text is reached in normal progression, I then fill in the interestices left by the earlier brief exposure. By no later than the middle of the semester—say at about Chapter 6—I am able to integrate directly with the text the introduction of language structures necessary for problems.

One of the acknowledged difficulties of teaching the first course in computer science is the problem of how to handle input and output. Not only are the language structures for input and output the most difficult parts of most languages, but they are surely the most tedious. I am thoroughly convinced that the beginning student should use one of the "free-format" or related types of input-output facilities which, while not officially part of all procedure-oriented languages, are available at almost all installations. Formatted output should not be introduced until nearly the end of the course. Moreover, when it is introduced, it should not be with stultifying detail. This is the prime reason why the discussion of formatted input and output in Chapter 10 is quite brief.

One alternative to free-format input and output is an automatic grading system, types of which are in use at a number of universities, in which all input and output is done by the system for the student. Such systems have various merits but they do tend to give students an unrealistic impression of how computing really gets done and, in particular, the student using such a system often gets no experience in developing the very important skill of generating good test data.

NOTES FOR INSTRUCTORS

Starred sections (*) in the Contents can be omitted without loss of continuity. Most of these contain topics outside the main theme of computer

language, but it would seem to me to be unwise to omit all such sections. Daggered sections (†) contain material I consider of direct importance to users of procedure-oriented languages but, nevertheless, these can be omitted if lack of time necessitates some compromises. The following comments may also be useful in planning a course.

Chapter 1 contains introductory material meant to be read rather than lectured from; most instructors will probably wish to give their own introductory lecture or two. Chapter 6 is another likely candidate for reading without lecturing since most of it should be partially familiar to students by the time it is reached in normal sequence through the book. Finally, Chapter 11, intended mainly as a bridge to later courses, can be assigned for reading if insufficient time remains to discuss it in lectures. This chapter is a purposely sketchy introduction to operating systems and time sharing. One reason for this sketchiness is the diversity of computing milieus which exist for students taking a first course in computers. Another is that, by the time the end of the course is reached, many students will have developed a feeling for the topics in this chapter from their experience in the course and because many instructors will slip information on the topics discussed in Chapter 11 into their lectures as the course progresses.

Although little or no mathematical background is required for a course from this book, some instructors will wish to omit those sections which emphasize mathematics. These include Sections 3.1.1 through 3.1.5 and Section 5.1 which emphasize arithmetic concepts, Section 5.3 on compilation of arithmetic expressions, which emphasizes Polish notation, and Sections 9.1 and 9.2 on Boolean algebra and logical design.

One departure from the more usual order of introduction of topics is the relatively late (Chapter 8) discussion of language structures for iteration. There is no intention in this to downgrade the importance of repetitive calculation using different sets of data. No idea is more important for the beginning student and, indeed, this idea is introduced much earlier than Chapter 8. Rather, my attitude concerns the importance of distinguishing between language structures which really extend the capabilities of the language and those which merely provide added convenience to the users. Usually, there is too little stress placed on this point. Language structures for iteration are a particularly obvious case of an addition of convenience (an important convenience, of course), but not of capability. If one takes the point of view that the order of introduction of topics should facilitate the assignment of computer projects involving algorithms of increasing complexity, then postponing the introduction of iteration structures until relatively late is easily defensible.

To conclude these notes, I offer two outlines of courses including topics and a suggested number of lectures, one for a basic course emphasizing only language and another for a more sophisticated course. As an alternative to the latter, a full year, in-depth course could be taught from this book.

Basic Course		More Advanced Course	
Chapter or Section	*Number of Lectures*	*Chapter or Section*	*Number of Lectures*
1	3	1	2
2	4	2	3
3.2, 3.2.1	1	3.1	3
4.1	3	3.2	2
4.3	2	4.1, 4.2, 4.3	3
4.4, 4.4.1, 4.4.2	2	4.4	2
5.2	3	5.1	1
5.4, 5.5	4	5.2	2
6	3	5.3	2
7.1, 7.2	2	5.4, 5.5	3
7.3	4	6	2
7.4.1, 7.5	1	7.1, 7.2	1
8.1, 8.2	4	7.3	2
9.3	2	7.4, 7.5	1
10.2, 10.3	2	8.1, 8.2	3
	40	8.3	1
		9.1, 9.2	3
		9.3	1
		10.1	2
		10.2, 10.3	2
		11	2
			43

The remaining periods might be used for examinations and topics of the instructor's choice.

ACKNOWLEDGMENTS

The variety of topics and number of computer languages discussed in this book mean that I am more than usually indebted to those of my colleagues who have read all or part of the manuscript of this book and have helped me avoid numerous errors. (Those that remain are, of course, all my own responsibility.)

In particular I must thank Alan Perlis, Robert Rosin, Phyllis Fox, Erich Schmitt, Gilbert Berglass, Albert Allan, Joel Herbsman, Richard Eckhouse, and William Fredson-Cole. In addition, I must thank two years of students at the State University of New York at Buffalo who suffered through various versions of the manuscript of this book and found errors in it and made comments on it. Finally, I must thank—although mere thanks are not really enough—the legion involved in typing the manuscript, particularly, Rita Keller, Deborah Finn, Joyce Staskiewicz, Lynn Fagyas, Linda Janos, and my wife.

Anthony Ralston

conventions, notation, and abbreviations

CONVENTIONS

One of the problems in a book largely concerned with computer programming is the conventions to be used in printing programs and, particularly, program fragments in the text itself. This is especially true when, as here, more than one programming language is being discussed. The conventions we have used, hopefully consistently applied, are as follows:

Fortran and Cobol—Upper case used throughout, not only in programs, but also when naming statements in text (e.g., "An EQUIVALENCE statement is . . . .").

Algol—The publication language using boldface keywords is used throughout in programs and text, except that semicolons after statements are omitted in the text (e.g., "The Algol statement A := B[6] := C[I,J] is an example of . . . .").

PL/I—The 60-character set is used in all programs and text; as with Algol, semicolons are omitted after statements in text.

NOTATION

Syntactical Notation

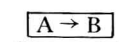
This notation, which is introduced on page 34, is as follows:

<---> to be read as "The structure (whatever is contained in the brackets) of the language"

::=	to be read as "is defined to be"
|	to be read as "or"
$[n]$	these brackets placed over the symbol ::= with an integer n inside them indicate that n is the maximum number of symbols allowed in the definition which follows ::=

The quantity in <———> will always be upper case letters. For the sake of informality we shall sometimes replace the <———> with lower case italics (e.g., <LETTER> → *letter*).

General Notation

Symbol or Example	*Meaning*	*First used on page*
(Problem 14)	References to problems at the end of each chapter	40
{ *vn* | (<LIST>) }	Choice in syntax—quantities in braces, one of which must be chosen	199
[*label*]	Optionality in syntax—quantity in brackets which may be omitted	319
L(A)	Location of quantity in computer memory—L followed by name of quantity in parentheses	169
$(101.100)_2$	Base of number system—subscript following parenthesized number	68
$\sum_{i=1}^{n}$	Summation—Greek sigma with limits of summation above and below	70
	Logical symbols:	
$\wedge$ or ·	And	357
$\vee$ or +	Or	357
$\neg$ or -- (overbar)	Not	357
$\equiv$	Equivalence	375
$\supset$	Implication	375
	Flow chart symbols:	
	Function box	36
	Decision box	37
	Remote connector	37

ABBREVIATIONS

In the panels which consider language structures in various computer languages, a number of abbreviations are used for conciseness. These are always explained on the panel itself but, since abbreviations are also used occasionally in the text, we give a complete list here:

ae—arithmetic expression
ar—argument
de—designational expression
exp—expression
fn—format number
id—identifier
int—integer
ip—increment part
iv—integer variable
le—logical expression
lv—logical variable
pint—positive integer
pn—procedure name
re—relational expression
ro—relational operator
sl—statement label
slv—statement label variable
st—statement (executable)
sv—scalar variable
td—type declaration
un—unit number
var—variable