Texts and Monographs in Computer Science

Hans W. Gschwind and Edward J. McClusky
Design of Digital Computers
An Introduction
2nd Edition. 1975. viii, 548p. 375 illus. cloth

Brian Randell, Ed.
The Origins of Digital Computers
Selected Papers
2nd Edition. 1975. xvi, 464p. 120 illus. cloth

Jeffrey R. Sampson
Adaptive Information Processing
An Introductory Survey
1976. x, 214p. 83 illus. cloth

Arto Salomaa and M. Soittola
Automata-Theoretic Aspects of Formal Power Series
1978. x, 171p. cloth

Saud Alagic and Michael A. Arbib
The Design of Well-Structured and Correct Programs
1978. x, 292p. 68 illus. cloth

Michael A. Arbib, A.J. Kfoury, and Robert N. Moll
A Basis for Theoretical Computer Science
1981. viii, 220p. 49 illus. cloth

The Science of Programming

David Gries

Springer-Verlag
New York Heidelberg Berlin

David Gries
Department of Computer Science
Cornell University
Upson Hall
Ithaca, NY 14853
U.S.A.

Library of Congress Cataloging in Publication Data
Gries, David, 1939–
The science of programming.
(Texts and monographs in computer science)
Bibliography: p.
Includes index.
1. Electronic digital computers—Programming.
I. Title. II. Series.
QA76.6.G747 001.64'2 81-14554
AACR2

Printed in the United States of America.

9 8 7 6 5 4 3 2 1

ISBN 0-387-90641-X Springer-Verlag New York Heidelberg Berlin
ISBN 3-540-90641-X Springer-Verlag Berlin Heidelberg New York

Foreword

This is the textbook I hoped someone like Professor David Gries would write —and, since the latter has no rivals, that means I just hoped he would write it. The topic deserves no lesser author.

During the last decade, the potential meaning of the word "program" has changed profoundly. While the "program" we wrote ten years ago and the "program" we can write today can both be executed by a computer, that is about all they have in common. Apart from that superficial similarity, they are so fundamentally different that it is confusing to denote both with the same term. The difference between the "old program" and the "new program" is as profound as the difference between a conjecture and a proven theorem, between pre-scientific knowledge of mathematical facts and consequences rigorously deduced from a body of postulates.

Remembering how many centuries it has taken Mankind to appreciate fully the profundity of this latter distinction, we get a glimpse of the educational challenge we are facing: besides teaching technicalities, we have to overcome the mental resistance always evoked when it is shown how the techniques of scientific thought can be fruitfully applied to a next area of human endeavour. (We have already heard all the objections, which are so traditional they could have been predicted: "old programs" are good enough, "new programs" are no better and are too difficult to design in realistic situations, correctness of progams is much less important than correctness of specifications, the "real world" does not care about proofs, etc. Typically, these objections come from people that don't master the techniques they object to.)

It does not suffice just to explain the formal machinery that enables us to design "new programs". New formalisms are always frightening, and it takes much careful teaching to convince the novice that the formalism is

not only helpful but even indispensable. Choice and order of examples are as important as the good taste with which the formalism is applied. To get the message across requires a scientist that combines his scientific involvement in the subject with the precious gifts of a devoted teacher. We should consider ourselves fortunate that Professor David Gries has met the challenge.

Edsger W. Dijkstra

Preface

The *Oxford English Dictionary* contains the following sentence concerning the term *science*:

> Sometimes, however, the term *science* is extended to denote a department of practical work which depends on the knowledge and conscious application of principles; an art, on the other hand, being understood to require merely knowledge of traditional rules and skill required by habit.

It is in this context that the title of this book was chosen. Programming began as an art, and even today most people learn only by watching others perform (e.g. a lecturer, a friend) and through habit, with little direction as to the principles involved. In the past 10 years, however, research has uncovered some useful theory and principles, and we are reaching the point where we can begin to teach the principles so that they can be consciously applied. This text is an attempt to convey my understanding of and excitement for this just-emerging science of programming.

The approach does require some mathematical maturity and the will to try something new. A programmer with two years experience, or a junior or senior computer science major in college, can master the material —at least, this is the level I have aimed at.

A common criticism of the approach used in this book is that it has been used only for small (one or two pages of program text), albeit complex, problems. While this may be true so far, it is not an argument for ignoring the approach. In my opinion it is the best approach to reasoning about programs, and I believe the next ten years will see it extended to and practiced on large programs. Moreover, since every large program consists of many small programs, it is safe to say the following:

> One cannot learn to write large programs effectively until
> one has learned to write small ones effectively.

While success cannot be guaranteed, my experience is that the approach often leads to shorter, clearer, *correct* programs in the same amount of time. It also leads to a different frame of mind, in that one becomes more careful about definitions of variables, about style, about clarity. Since most programmers currently have difficulty developing even small programs, and the small programs they develop are not very readable, studying the approach should prove useful.

The book contains little or no discussion of checking for errors, of making programs robust, of testing programs and the like. This is not because these aspects are unimportant or because the approach does not allow for them. It is simply that, in order to convey the material as simply as possible, it is necessary to concentrate on the one aspect of developing correct programs. The teacher using this book may want to discuss these other issues as well.

The Organization of the Book

Part I is an introduction to the propositional and predicate calculi. Mastery of this material is important, for the predicate calculus should be used as a *tool* for doing practical reasoning about programs. Any discipline in which severe complexity arises usually turns to mathematics to help control that complexity. Programming is no different.

Rest assured that I have attempted to convey this material from the programmer's viewpoint. Completeness, soundness, etc., are not mentioned, because the programmer has no need to study these issues. He needs to be able to manipulate and simplify propositions and predicates when developing programs.

Chapter 3, which is quite long, discusses reasoning using a "natural deduction system". I wrote this chapter to learn about such systems and to see how effective they were for reasoning about programs, because a number of mechanical verifiers systems are based on them. My conclusion is that the more traditional approach of chapter 2 is far more useful, but I have left chapter 3 in for those whose tastes run to the natural deduction systems. Chapter 3 may be skipped entirely, although it may prove useful in a course that covers some formal logic and theory.

If one is familiar with a few concepts of logic, it is certainly possible to begin reading this book with Part II and to refer to Part I only for conventions and notation. The teacher using this text in a course may also want to present the material in a different order, presenting, for example, the material on quantification later in the course when it is first needed.

Part II defines a small language in terms of weakest preconditions. The important parts —the ones needed for later understanding of the development of programs— are chapters 7 and 8, sections 9.1 and 9.2, and chapters 10 and 11. Further, it is possible to skip some of the material, for example the formal definition of the iterative construct and the proof of theorem 11.6 concerning the use of a loop invariant, although I believe that mastering this material will be beneficial.

Part III is the heart of the book. Within it, in order to get the reader more actively involved, I have tried the following technique. At a point, a question will be raised, which the reader is expected to answer. The question is followed by white space, a horizontal line, and more white space. After answering the question, the reader can then continue and discover my answer. Such active involvement will be more difficult than simply reading the text, but it will be far more beneficial.

Chapter 21 is fun. It concerns inverting programs, something that Edsger W. Dijkstra and his colleague Wim Feijen dreamed up. Whether it is really useful has not been decided, but it is fun. Chapter 22 presents a few simple rules on documenting programs; the material can be read before the rest of the book. Chapter 23 contains a brief, personal history of this science of programming and an anecdotal history of the programming problems in the book.

Answers to some exercises are included —all answers are not given so the exercises can be used as homework. A complete set of answers can be obtained at nominal cost by requesting it, on appropriate letterhead.

Notation. The notation *iff* is used for "if and only if". A few years ago, while lecturing in Denmark, I used *fif* instead, reasoning that since "if and only if" was a symmetric concept its notation should be symmetric also. Without knowing it, I had punned in Danish and the audience laughed, for *fif* in Danish means "a little trick". I resolved thereafter to use *fif* so I could tell my joke, but my colleagues talked me out of it.

The symbol □ is used to mark the end of theorems, definitions, examples, and so forth. When beginning to produce this book on the phototypesetter, it was discovered that the mathematical quantifiers "forall" and "exists" could not be built easily, so $\boldsymbol{A}$ and $\boldsymbol{E}$ have been used for them.

Throughout the book, in the few places they occur, the words *he*, *him* and *his* denote a person of either sex.

Acknowledgements

Those familiar with Edsger W. Dijkstra's monograph *A Discipline of Programming* will find his influence throughout this book. The calculus for the derivation of programs, the style of developing programs, and many of the examples are his. In addition, his criticisms of drafts of this book have been invaluable.

Just as important to me has been the work of Tony Hoare. His paper on an axiomatic basis for programming was the start of a new era, not only in its technical contribution but in its taste and style, and his work since then has continued to influence me. Tony's excellent, detailed criticisms of a draft of Part I caused me to reorganize and rewrite major parts of it.

I am grateful to Fred Schneider, who read the first drafts of all chapters and gave technical and stylistic suggestions on almost every paragraph.

A number of people have given me substantial constructive criticisms on all or parts of the manuscript. For their help I would like to thank Greg Andrews, Michael Gordon, Eric Hehner, Gary Levin, Doug McIlroy, Bob Melville, Jay Misra, Hal Perkins, John Williams, Michael Woodger and David Wright.

My appreciation goes also to the Cornell Computer Science Community. The students of course CS600 have been my guinea pigs for the past five years, and the faculty and students have tolerated my preachings about programming in a very amiable way. Cornell has been an excellent place to perform my research.

This book was typed and edited by myself, using the departmental PDP11/60-VAX system running under UNIX* and a screen editor written for the Terak. (The files for the book contain 844,592 characters.) The final copy was produced using *troff* and a Comp Edit phototypsetter at the Graphics Lab at Cornell. Doug McIlroy introduced me to many of the intricacies of *troff*; Alan Demers, Dean Krafft and Mike Hammond provided much help with the PDP11/60-VAX system; and Alan Demers, Barbara Gingras and Sándor Halász spent many hours helping me connect the output of *troff* to the phototypesetter. To them I am grateful.

The National Science Foundation has given me continual support for my research, which led to this book.

Finally, I thank my wife, Elaine, and children, Paul and Susan, for their love and patience over the past one and one half years.

*UNIX is a trademark of Bell Laboratories.

Table of Contents

1 Cries Pt 1
2 " Ch 6, 7, 8
3 " " 9
4 " " 10
~~5 " " "~~
5 " " 11
6 " " 11
7 " " 16
8 " " 16, 17, Dromey
9 " " 19
10 " " "