Decker & Hirshfield

The Analytical Engine

An Introduction to Computer Science Using HyperCard 2.1

Second Edition

SECOND EDITION

THE ANALYTICAL ENGINE

AN INTRODUCTION TO COMPUTER SCIENCE USING HYPERCARD 2.1

RICK DECKER
STUART HIRSHFIELD
Hamilton College

PWS PUBLISHING COMPANY
BOSTON

PWS Publishing Company
20 Park Plaza, Boston, MA 02116-4324

Library of Congress Cataloging-in-Publication Data

Decker, Rick.
The analytical engine: an introduction to computer science using Hypercard 2.1 / Rick Decker, Stuart Hirshfield—2nd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-534-93696-2
1. Electronic digital computers. 2. Macintosh (Computer)—Programming. 3. Hypercard (Computer file). I. Hirshfield, Stuart. II. Title.
QA76.5.D37 1994 93-25430
004—dc20 CIP

Sponsoring Editor: Michael J. Sugarman
Associate Developmental Editor: Susan M. Gay
Editorial Assistant: Ken Morton
Production Editor: Abigail M. Heim
Manufacturing Coordinator: Ruth Graham
Interior Designer: Abigail M. Heim
Cover Designer: Sally Bindari, Books by Design
Compositor: Pure Imaging
Cover Printer: John P. Pow Co., Inc.
Text Printer and Binder: Courier/Westford

Printed and bound in the United States of America
94 95 96 97 98—9 8 7 6 5 4 3 2 1

PREFACE

THE Æ COURSE

What you have in your hands represents a departure from the traditional introduction to computer science textbooks. Indeed, we have coined the term "CS 0" to describe the course that this text/disk package embodies. Because this package is different, a few words of introduction and encouragement are in order.

We wrote the text and prepared the lab disks for the same reason we suspect many other authors do: We simply couldn't find an existing package that suited our needs. As at many other schools, our introductory course was a programming course that served two audiences: those who wanted an introduction to the subject and those who intended to major in computer science (or at least take some courses beyond the introductory level). And, as at many other schools, this approach simply didn't work well. Because of the wide range of talents and backgrounds of the students we were constantly performing a balancing act, trying to move slowly enough not to lose the bottom half of the class and quickly enough not to bore the students with some prior experience. The bimodal nature of the course was difficult to deal with, both for us and the students. Even more problematic was our perception that students were finishing the introductory course with the firm impression that computer science was nothing but programming. This attitude did a disservice to those wanting a taste of the discipline and frequently left our majors in shock when they later discovered that being a programming whiz counted for little in their subsequent courses.

"You get too soon old and too late smart," the saying goes. Having taught a programming introductory course for seven years, we finally realized that something had to be done and, more importantly, we realized that what had to be done was something that hadn't been done before, at least not in computer science. Consider introductory courses in other disciplines—English 101 does not consist solely of having the students complete writing exercises; of course, the students typically have a number of essays to write, but they are also exposed to the broad historical trends in literature, they are introduced to the forms of literature, and they are given the critical apparatus necessary to make sense of the material they read. The students enrolled in Physics 101 are likewise learning to manipulate laboratory equipment while being exposed to the material in context—perhaps learning about Galileo, Maxwell, and Einstein, certainly exploring the major divisions of the subject, and being exposed to the social, political, and moral implications of the use and misuse of physical discoveries.

Introductory courses in computer science, on the other hand, typically tend to suffer from one or more major shortcomings:

- *Equating computer science with programming.* One of the things we hear again and again from our students is the mistaken idea that computer science is programming. Of course, computer professionals do write programs from time to time. Computer science, though, steps back from programming and, like physics, seeks to formulate and understand the general principles that govern the objects of its study, which for us are computers and their programs. The study of computer science is related to writing programs in somewhat the same way that the study of music is related to the production of songs. The product is important, but the study of the principles behind the product is vastly more so: It is nearly impossible to produce the product without some understanding of the principles.
- *Confusing training with education.* Another popular form of the introductory course is what we might call "Getting Acquainted With 4th Dimension, WriteNow, and Excel." A slightly more elevated version of this course also exists in the form "So You Want a Career in MIS?" Neither version has much to do with the discipline of computer science, and, given the rapid advances in the field, both run the risk of providing specific training in technologies that will be out of date by the time the students graduate.
- *Concentrating on effects at the expense of causes.* In an attempt to avoid alienating their audience by introducing technical material, some introductory courses sidestep computer science almost entirely, stressing instead the social consequences of the computerization of society. Done poorly, such a course can become what one of our colleagues calls the "*People* Magazine Goes to MIT" approach. Done well, though, this approach can be valuable. We feel that it is important for every citizen to be aware of the possible consequences of the use of technology, but we also feel that to understand the implications of technology it is necessary to understand the technology itself. We believe that along with questions of what computers should and should not do, our students should also be aware of what they *can* and *cannot* do, both by virtue of the current state of the art and theoretical limitations.

We set out to design a true survey course, presenting a serious disciplinary point of view, firmly grounded in a liberal arts tradition. The collective experience of the authors (we have taught this course for five years now), our students, and our many faithful adopters seems to indicate that this approach—the Æ approach to CS 0—overcomes the aforementioned shortcomings while serving all of the course's constituencies.

THE TEXT

This second edition retains the basic organization and outline of the first. The arrangement of the topics proceeds first downward, to increasingly concrete points of view, and then up, returning to increasingly more general levels of abstraction—a kind of *Divine Comedy* itinerary.

Module 1 provides a historical orientation, describing the technological history of computers in the context of increasing use of technology, beginning with the Industrial Revolution. The lab portion of this module is devoted to an introduction to the Macintosh and HyperCard. **Module 2** discusses some computer applications—the familiar (calculators, word processors, and spreadsheets) as well as some more specialized and less familiar ones in medicine, the sciences, and education. This module concludes with an introduction to some social implications of computer use, a theme that is continued in Module 9. The lab part of this module provides the students with hands-on experience with HyperCard implementations of a simple word processor, a spreadsheet, a graphing calculator, a DNA pattern-matching stack, and arithmetic flash cards.

Modules 3 through 6 lead the students deeper into the inner circles of the abyss. **Module 3** discusses system design, using the example of the user interface. At this level the focus is on combining components with fully developed functionalities into a smoothly functioning system: A program begins to become less of a "black box" and the details begin to be apparent. The lab part of Module 3 is devoted to the authoring level of HyperCard; the students have a practice stack that provides a tutorial on stack design, fields, and buttons. The lab concludes with a restaurant guide stack that the students are asked to customize.

In **Module 4** the gray box becomes a clear box: Students are introduced to programming by inspecting scripts of existing stacks and writing scripts of their own. We discuss most of the canonical programming constructs as well as algorithm design, and we take students through a simple software life cycle, using the lab stack as an example. The lab portion of this module provides an accounting application with sorting and searching capabilities. The students are directed to modify and expand this stack.

Module 5 deals with program translation. The important idea here, of course, is that since a computer can only execute programs in its own machine language, a source program in HyperTalk must be translated into machine language to be executed. We discuss the problem of representing text in binary form and provide an assembler for a hypothetical computer. The labs for this module follow the pattern of all subsequent labs: Now that the students have been introduced to programming, they not only can run the labs to reinforce the text material, but they can also inspect and modify the scripts of the lab stacks. The first lab stack is a text-to-ASCII-to-binary converter, and the second is an assembler for the simulated computer.

Module 6 concludes the progress toward the concrete by describing how the hardware of a computer works. Starting with switches, we construct gates, which we combine to construct logic, arithmetic, and memory circuits. Finally, we use the circuits to build the small computer that was only hypothetical in Module 5. The lab stack for this module is a simulated breadboard that the students can use to design and test circuits of their own.

Modules 7, 8, and 9 ascend from the most concrete, physical level to the most abstract and general. In **Module 7** we make two points: First, that before there were physical realizations of computers, there were abstract, mathematical ones; second, that the physical machine is in some sense nonessential when thinking about the nature of programs and computation. We introduce the Turing Machine, discuss the ideas of encoding strings and programs, and show that there are infinitely many tasks that computers cannot do, not only because there are uncountably many input-output matchings and only countably many programs, but also because there are tasks (like the Halting Problem) that seem natural candidates for computer solution but are simply impossible to program. The lab for Module 7 is a Turing Machine simulator.

Module 8 is a segue, via Turing, from questions of what computers cannot do to what they might do. We use Arthur C. Clarke's HAL 9000 computer as a standard against which we view the current state of affairs in artificial intelligence research. The lab stacks include a poetry generator and a simulated optical character recognizer.

Finally, in **Module 9** we look at things to come. We identify the major trends in computer use and try to see what the implications of these trends might be, guided at all times by a knowledge of how difficult it is to predict the future. The lab stack, an ATM simulation, serves to demonstrate the basic concepts of security, privacy, and maintenance as they apply to computer systems and networks.

NEW TEXT FEATURES FOR THE SECOND EDITION

While the topical organization of the text has remained intact from the first edition, this second edition is indeed "new and improved" in a number of important ways. The changes we chose to incorporate reflect the expressed preferences of our "users"—that is, the students and faculty who have used the package in the classroom. In a nutshell, the new features of the text include:

- *Lab exercises that are "folded into" the text material for each module:* The first edition had this feature in only two of its nine modules.

It worked so well in those modules to better integrate the lab and the text (and also to break up long blocks of text) that we decided to do it throughout the book.

- *More detailed lab exercises, many involving writing:* One by-product of "folded-in" lab exercises is that many of the exercises could be rewritten to concentrate on particular sections of the text. As a result, the exercises are much more thorough and relevant to the text. Also, a number of new exercises have been written that ask students to write out English comments about particular lab experiences.
- *A textual version of balloon help:* In the dual interests of highlighting points made in the text and making it easier to find subsections of the text, we have added marginal notes (in the form of "balloon help," à la System 7).
- *No presumptions about computing environment:* The primary motivation for including a System Folder and HyperCard with the first edition software was to provide students with a "turnkey" environment. We now recognize (thanks to our adopters!) that this was a mistake. Every user of the package operates in an ever-so-slightly different environment with a complex combination of machine models, disk drives, system folders, and network connectivity. The new edition of the text is, as a result, decidedly less prescriptive about such matters. For example, the Quit button that appears on all Æ stacks no longer shuts the machine (and potentially your network!) down, but rather politely quits HyperCard and returns to the Mac desktop.

SUPPLEMENTARY MATERIALS

The *Instructor's Manual* includes transparency masters, and may be ordered (by instructors only) separately or with a *Sample Student Program Disk,* which contains stacks created by student users of the text.

THE Æ STACKS

This is a lab-based course. It might be possible to offer this course without a lab component, but we think it would be a serious mistake to do so. Computer science, like the other physical sciences, is a lab science. It is also a contact, rather than a spectator, sport. The disk that comes with the text contains all of the HyperCard stacks referenced in the text and lab modules. As long as you have access to HyperCard (version 2.0 or higher), and a Macintosh that will run it, you are in business.

The lab-based nature of the course was dictated by our experience with computer labs in an introductory computer science course at Hamilton College over the past ten years. We could talk forever about our discipline, but the best way for you to understand what we're talking about is to have hands-on experience, the more the better.

Interwoven with every text module, in addition to a variety of pencil and paper (keyboard and screen?) exercises, is a collection of directed lab exercises. These exercises are based on the disk materials that accompany the text and are central to the course. In the process of accomplishing the exercises, students will experience first-hand a word processor, a spreadsheet, a database system, a logic breadboard, an assembler, a Turing Machine simulator, a poetry generator, and more. All of these programs were designed to support the text directly, and all were written using HyperCard.

Note: See the Appendix for a detailed description of the Analytical Engine disks, its setup and use.

Why HyperCard, you ask? In our opinion, HyperCard is the first commercially available program to offer software capable of supporting a true survey course. First and foremost is the fact that HyperCard provides a medium in which students can use the computer in interesting and creative ways *without being programmers.* Simply by learning how to navigate through and edit HyperCard stacks, one can develop an appreciation for how computer applications and languages work, how they are designed, and how they interact to form computer systems. The applications that the students use, design, and edit can then be examined from the perspective of programming by clicking the Macintosh mouse to examine the underlying programs. This approach is in marked contrast to the heretofore standard model that required students to spend an entire semester learning to write a program to perform a calculation that they could have solved in a few minutes using paper and pencil.

Programming a computer provides students with many valuable insights into how a computer works and how a computer scientist thinks. After all, if computer science is concerned in part with the study of programs, as we've said, what better way to begin than by having our students write some programs on their own? Our experience has been that for the audience of a survey course, where we assume no prior programming experience, the programming process itself is tough enough without having to master the mass of syntactic details of a language. Along with its many other capabilities, HyperCard includes a programming language, HyperTalk, which is distinguished by its very natural syntax. HyperTalk programs read almost like collections of English statements, enough so that if you leave out a word, HyperCard has at least a fighting chance of figuring out what you meant and instructing the Mac to do it.

NEW SOFTWARE FEATURES FOR THE SECOND EDITION

Almost by definition, software becomes outdated as soon as it is released, and the collection of Æ stacks that accompanied the first edition was no exception. While we were (and remain) quite pleased with the original text, we knew that the first edition of the software could be improved. The fact that the original software won an award from EDUCOM for curricular innovation has not stopped us from making significant revisions and improvements, as follows:

- *All Æ stacks (and associated lab exercises) have been rewritten from scratch to take advantage of HyperCard version 2.1* (but can be used with version 2.0).
- *Functionality and error-checking have been improved for many of the original stacks.* For example, spreadsheets and Logg-O circuits can now be saved to and retrieved from disk, Logg-O now accommodates combinational circuits, and all stacks that create data files also now create file signatures that ensure that only appropriate data can be opened.
- *The less inspired of the original stacks have been eliminated and replaced with more motivating, better-implemented ones.* Gone are Æpplications, Calendar, Practice, Little Mac Book, DR, and DR No!. Newly created, and supported by lab exercises, are GraphiCalc (a graphing calculator), Fitted Genes (the aforementioned DNA pattern-matcher), Flasher! (the aforementioned flash cards), Bill's Diner (a restaurant guide), Æ Workbook (an updated version of the original Practice stack), HyperChars (an OCR simulation), and ÆTM (a banking machine).
- *New utility-like stacks have been written.* These introduce and provide students with the ability to incorporate animation (Ært Show) and Macintosh resources (sounds, icons, and cursors, in stack Very Resourceful) into their stacks.

SCOPE AND ORDER OF TOPICS

We have made some very deliberate choices in choosing material for and organizing this text. Even a casual review of the table of contents gives the impression that the text covers a great deal of material. It does! One of our early decisions was to commit errors of commission, as opposed to those of omission. To be sure, there is more material in our text than can be covered in a semester course at most schools—including our own. (As indicated by the sample syllabus below, our version of the course pays only casual attention to many of the topics [Module 7, for example] and ignores others altogether [the Pip material in Modules 5 and 6]. This reflects both the interests of our audience and our curriculum.) On the other hand, we have not devoted entire modules to specific "hot" computer applications (except, of course, HyperCard). We have included what we regard as the core material of the discipline—material that is principled and resistant to change—and there is a lot of it.

The order of presentation of the topics reflects the lab orientation of the course. We want students to learn by doing, as well as by reading and thinking. The progression in Modules 2 to 6 from a black box, to a gray box, to a microscopic clear box, gives the students experience with a computer at a particular level of abstraction before taking them down to the next level. Having just used application stacks, students can customize them, evaluate user interfaces, and design their own. Having just designed a stack, they can click on a button and see the programs that underlay it. After using HyperTalk, students wonder how it is that the computer understands such a high-level language. The "language" that the machine understands is logic, and the Module 6 lab convinces most students that logical devices can be built to accomplish a number of interesting tasks. Having seen how the machine does what it does, and with a base of practical experience, it is then appropriate to question, as we do in Modules 7 to 9, the machine's theoretical limitations, the current boundaries of the discipline, and the social implications of the technology.

Also as a result of the lab orientation, this text is more tightly structured than many others. Using "depends on the material from" as a relation on the set of modules in this text, we find that the text is linearly ordered. We know that our order of presentation is not the one everyone would use, and we make no apologies about that. You can teach program translation after hardware or reverse the order of presentation of the entire text, if you wish, but be aware that in doing so you run the risk of dangling forward references. If you have good luck with a different order, let us know.

Hamilton College has 14-week semesters. Our syllabus for this course looks like this:

Module 1:	2 lectures, 1 lab	include a Mac tour for novices
Module 2:	2 lectures, 1 lab	demonstrate a variety of Mac applications
Module 3:	4 lectures, 2 labs	use the lab stacks to discuss design
Module 4:	4 lectures, 2 labs	review scripts of familiar stacks
Module 5:	3 lectures, 2 labs	(Pip material is not covered)
Module 6:	3 lectures, 2 labs	
Module 7:	2 lectures, 1 lab	use lab stack in casual, high-level coverage
Module 8:	3 lectures, 1 lab	show *2001: A Space Odyssey*
Module 9:	2 lectures, 1 lab	

This syllabus provides 38 class meetings, leaving the rest for exams, additional lab sessions, and supplemental material (including scores of relevant films and tapes). Of special note is the series of video tapes entitled "The

Machine That Changed the World," aired on PBS and produced in part by the ACM. The five tapes in the series (available as a set at a very reasonable cost) fit almost perfectly with our modules 1, 2, 3, 8, and 9, respectively, and help to bring the associated topics alive for our students.

PERORATION

Although this project was in many ways our creation, it would not exist in its present form without the contributions of many talented and dedicated people, each of whom influenced the final product in some significant and positive way. Our thanks go out to the following people for their insightful reviews of the original manuscript: Professors Dwight Barnette, Virginia Polytechnic Institute; Lee Bryant, SUNY, Geneseo; Scott Drysdale, Dartmouth College; Jim Gips, Boston College; Gordon Goodman, Rochester Institute of Technology; Will Goodwin, University of Oregon; Dan Kimura, George Washington University; Joan Krone, Ohio State University; Curt Lauckner, Eastern Michigan University; Henry Leitner, Harvard University; Jeff Naughton, Princeton University; Jeff Parker, Boston College; Ellie Quinlan, Ohio State University; Allen Tucker, Bowdoin College; and Henry Walker, Grinnell College; and also to the reviewers of the second edition:

Anselm Blumer
Tufts University

Bill Chen
University of Hawaii at Hilo

Matthew Dickerson
Middlebury College

Batya Friedman
Colby College

Otto Hernandez
Atlantic Community College

Jacquelyn Jarboe
Boise State University

Lawrence S. Kroll
San Francisco State University

Kenneth L. Modesitt
Western Kentucky University

Joseph O'Rourke
Smith College

Barbara Boucher Owens
St. Edward's University

Jane M. Ritter
University of Oregon

Robert Roos
Smith College

Scott Smith
SUNY, Plattsburg

Peter Wegner
Brown University

Robert J. Wernick
San Francisco State University

The changes made to produce this second edition result in a package that is, we believe, significantly improved. The second edition is more contemporary, cleaner, better tuned to our students, and even more empowering for them than was the first. For these improvements, we are also deeply indebted to those instructors who have shared their Æ experiences with us, to our students of the past five years, and to Frank Ruggirello, Mike Sugarman, Susan Gay, J. P. Lenney, Nathan Wilbur, Helen Walden, Abby Heim, Liz Clayton, Ken Morton, and Ed Murphy for trusting us.

Rick Decker
Stuart Hirshfield

CONTENTS