




FUNDAMENTALS OF COMPUTING I



L O G I C
P R O B L E M S O L V I N G
P R O G R A M S
A N D C O M P U T E R S

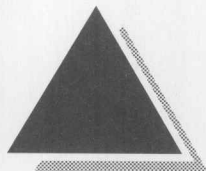


ALLEN B. TUCKER
W. JAMES BRADLEY
ROBERT D. CUPPER
DAVID K. GARNICK



USE
ITY OF

FUNDAMENTALS OF COMPUTING I



L O G I C

PROBLEM SOLVING

P R O G R A M S

AND COMPUTERS



ALLEN B. TUCKER

BOWDOIN COLLEGE

W. JAMES BRADLEY

CALVIN COLLEGE

ROBERT D. CUPPER

ALLEGHENY COLLEGE

DAVID K. GARNICK

BOWDOIN COLLEGE

McGRAW-HILL, INC.

NEW YORK ST. LOUIS SAN FRANCISCO AUCKLAND BOGOTÁ CARACAS
LISBON LONDON MADRID MEXICO MILAN MONTREAL NEW DELHI
PARIS SAN JUAN SINGAPORE SYDNEY TOKYO TORONTO

FUNDAMENTALS OF COMPUTING I: LOGIC, PROBLEM SOLVING, PROGRAMS, AND COMPUTERS

Copyright © 1992 by McGraw-Hill, Inc. All rights reserved. Printed in the United States of America.
Except as permitted under the United States Copyright Act of 1976, no part of this publication may be
reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without
the prior written permission of the publisher.

1 2 3 4 5 6 7 8 9 0 DOC DOC 9 0 9 8 7 6 5 4 3 2 1

ISBN 0-07-065449-2

This editor was Eric M. Munson;
the production supervisor was Anthony DiBartolomeo.
The cover was designed by Joseph Piliero.
R. R. Donnelley & Sons Company was printer and binder.

Library of Congress Cataloging-in-Publication Data

Fundamentals of computing / Allen B. Tucker... [et al.].

p. cm. — (McGraw-Hill computer science series)

Includes bibliographical references and index.

Contents: v. 1. Logic, problem solving, programs, and computers.

ISBN 0-07-065449-2 (v. 1)

I. Electronic data processing.

I. Tucker, Allen B.

II. Series.

QA76.F815 1992

004—dc20

91-32782

ABOUT THE AUTHORS

Allen B. Tucker is Professor and Chair of the Computer Science Department at Bowdoin College; he has held similar positions at Colgate University and Georgetown University. He earned a BA in mathematics from Wesleyan University in 1963 and an MS and PhD in computer science from Northwestern University in 1970. Professor Tucker is the author or coauthor of several books and articles in the areas of programming languages, natural language processing, and computer science education. He recently served on the ACM Task Force on the Core of Computing and as co-chair of the ACM/IEEE-CS Joint Curriculum Task Force that developed the report *Computing Curricula 1991*. He is a member of ACM, IEEE-CS, CPSR, and the Liberal Arts Computer Science Consortium (LACS).

W. James Bradley is Professor of Mathematics and Computer Science at Calvin College. He graduated from MIT with a major in Mathematics in 1964 and completed a PhD in Mathematics from the University of Rochester in 1974. Professor Bradley also earned an MS in Computer Science from the Rochester Institute of Technology in 1982. He has authored papers in game theory and computer science curriculum, as well as an introductory text in discrete mathematics. His current scholarly interests are in formal methods in decision making, database systems, ethical and social issues in computing, and computer science education. Professor Bradley is a member of MAA, ACM, CPSR, and LACS.

Robert D. Cupper is Professor and Chair of the Department of Computer Science at Allegheny College. He received a BS from Juniata College and a PhD from the University of Pittsburgh. At Allegheny, Professor Cupper developed one of the first computer science major programs for a liberal arts college, a program that helped motivate the design of the liberal arts model curriculum in 1986. He has been an active member of ACM for several years, having served as chair of the Student Chapters Committee and as secretary-treasurer of the Special Interest Group on Computer Science Education (SIGCSE). Professor Cupper has written and spoken on the economics of computing, curriculum development, and program accreditation. He is a member of ACM and a co-founder of LACS.

David K. Garnick is Assistant Professor of Computer Science and Dana Faculty Fellow at Bowdoin College. He earned a BA in philosophy at the University of Vermont and a PhD in computer science at the University of Delaware in 1988. He has conducted research in the area of programming language design for distributed computing, and his current work and publications are in the area of heuristic algorithms and combinatorial optimization problems. He also has interests and publications in curriculum design, with an emphasis on the integration of writing throughout the curriculum. He is a member of ACM.

ABOUT THE AUTHOR

To Meg, Hope, Sandy, and Kathy

FOREWORD

What is the computing profession? What is the discipline of computing? Our answers to these basic questions strongly influence our approaches to the content of computing curricula, the mix between theory and practice, our selection of research questions, our relations with other disciplines, our responses to complaints, our practices of design, and much more.

When I investigated these questions with the ACM Task Force on the Core of Computer Science beginning in 1986, I settled on definitions that are now widely accepted: the computing profession is people who make their livelihoods by working with computers and the phenomena surrounding computers. The core phenomena include algorithms and data structures, programming languages, architecture, numerical and symbolic computation, operating systems, databases and information retrieval, software methodology and engineering, artificial intelligence, and human-computer communication. These phenomena all concern representations of the world and efficient algorithmic transformations of those representations — commonly called information processing. In Europe the intellectual side of the profession is called “informatics” and in the U.S.A. people are beginning to call it the “discipline of computing.”

By 1990 I had come to be profoundly dissatisfied with this definition. It offered no guidance on the relation of my research and teaching to the burgeoning world of people using computers in their daily work. It offered no sense of permanence, leaving me with the nagging question of whether the discipline is a fad that will one day be reabsorbed into applied mathematics or electrical engineering. My dissatisfaction drew me into the question, “What is a profession?”

Underlying every profession is a permanent domain of human concern and human breakdown. By permanent, I mean concerns that affect every human being throughout civilization. By breakdown, I mean events that interrupt the normal flow of actions or work; these events may be the unanticipated failure of some person or system to deliver an expected result, or they may be the unexpected appearance of new challenges. The profession is the people, technologies, institutions, and practices for taking care of people’s concerns and recurrent breakdowns in the domain.

Consider the medical profession as an example. Health is a permanent concern of human beings. Breakdowns in health are inevitable because

of disease, accident, or aging. Health care professionals take care of people's concerns and breakdowns in health and disease. Stethoscopes, X-rays, MRI scanners, surgical tools, heart-lung machines, and pacemakers are some of the technologies of the profession. Laboratories, hospitals, HMOs, and medical schools are some of the institutions of this profession. Licenses, basic texts, anatomy charts, and diagnostic and surgical procedures are some of the standard practices of the profession.

The legal profession, another example, deals with people's concerns and recurrent breakdowns about laws. These concerns are inevitable and permanent because we all live in societies with governments, constitutions, and laws. Lawyers, judges, and law enforcers are among the members of this profession. They do their work within the technologies, institutions, and standard practices of this profession.

Now consider our profession. Calculation and coordination of action are ongoing concerns of all human beings. We live in a world of information and numbers, much of which are processed by machines. We live in a world with ubiquitous telephones, near-ubiquitous fax, and burgeoning computer networks and databases, all of which permit extending the distance and time over which we can successfully coordinate actions. Nearly everyone in every developed country is affected by telecommunications and computers, which open up new business and political opportunities; leaders in underdeveloped countries are considering informational infrastructures as ways of accelerating their countries' entries into world markets. Computation has become indispensable to the daily practices of finance, engineering, design, science, and technology. Word processing, accounting, database, design automation, and report writing software impact every other profession. This world offers many new kinds of breakdowns, ranging from failures of computers and communications to the challenge to install software that improves an organization's productivity. The computing profession, by analogy with other professions, is the people, technologies, institutions, and standard practices that take care of people's concerns in the domain of information processing, computation, and coordination over networks of computers.

These concerns are bigger than are implied by the phrase "phenomena surrounding computers". These concerns include, as is commonly understood, the design and analysis of hardware and software to perform new functions or to perform old functions in new ways. But these also include the installation, configuration, and maintenance of computer systems within organizations. They include standards for communication and information exchange. They include privacy and integrity of conversations, files, and documents in networks of computers. They include working with the customer to design computer systems that support the work of the customer's organization. They include the historical context of computing and communications, as well as the shared values of the people in the professions that use computers and networks.

In other words, the concerns are not phenomena that surround computers. It is the other way around. The computers surround the concerns.

If computer scientists continue to talk in language focused on “phenomena surrounding computers”, they will find themselves increasingly disconnected from the concerns people have about information processing and communications. Those people will turn elsewhere to get the help they need. There will be a computing profession, but it won’t include computer scientists as an important and vital part.

There need be no incompatibility between computer science research and people’s concerns for information processing and communication. In fact, research is an essential part of every profession, for it is the practice of anticipating future breakdowns and future opportunities. What’s missing is the skill of articulating the connection between research and people’s concerns. In the medical profession, for example, there are plenty of esoteric, highly technical projects without an immediate payback. If one asks such a medical researcher why he’s doing what he’s doing, one is likely to get an answer like this: “Even though this stuff is pretty technical and hard to understand, if it works we’ll one day be able to cure Alzheimer’s disease.” A computer science researcher might respond with, “I’m studying this because it is considered to be an open question among computer scientists.” With such an answer it is no wonder that outsiders look elsewhere for the help they seek, and that the computer science researcher is left wondering whether anyone is interested.

This book is a sharp break with the tradition of treating the discipline of computing merely as a study of phenomena surrounding computers. Its authors offer an introduction not just to the discipline, but to the profession. There are three distinguishing aspects of this book.

First, throughout the book, the authors maintain awareness of the connection between the technologies of programming, machines, and networks and the human concerns that animate these technologies.

Second, the authors distinguish between theory and practice. They hold that both are essential for a professional computer scientist. The main text emphasizes the theory while the laboratory emphasizes the corresponding practices. Students who emerge from this form of study will find themselves with a great deal more practical competence than their colleagues who study under the traditional, theory-oriented curriculum. The renewed interest in practice does not detract from the rigor of the discipline. The authors advocate the formalisms and rigorous thinking needed to underpin the practices of good programming and design, and they boldly discuss the laws and professional standards that define the environment in which students of the discipline will one day work.

Third, the authors bring to clear view the three paradigms of thought that constitute the discipline: theory, abstraction, and design. The theory paradigm is rooted in the long tradition of mathematics and logic, whose legacy enables us to deal with complex and subtle algorithms. The abstraction paradigm is rooted in the long tradition of the scientific method, whose legacy enables us to formulate and test hypotheses about algorithms, machines, and models. The design paradigm is rooted in the long tradition of engineering, whose legacy enables us to design machines that calculate accurately and process information in all domains of human work. The authors argue that the professional computer scientist must become competent in all three modes of thought. We can, at last, put aside the debates of which tradition is the most fundamental and revel in the realization that our profession is a unique combination of the three.

Other authors may improve on what Tucker, Bradley, Cupper, and Garlick offer here. But those future authors will be followers of these four pioneers in the new approach to computing as a discipline and a profession.

Peter J. Denning

PREFACE

The traditional introductory-level undergraduate courses in computer science and computer engineering (known simply as the discipline of *computing*) have received serious scrutiny over the past few years. These courses have been criticized for establishing the false notion that *computer science* = *programming*, thus leaving students with an inadequate view of the richness of the discipline. Many educators agree that we need serious changes in the way we organize and teach these courses.

This text, together with its accompanying laboratory manual and software tools, is the first in a four-volume series that aims to address this problem. This series presents the fundamental aspects of the discipline in a distinctive way; it uses an approach to the introductory curriculum that is often called the “breadth-first approach.”

Overview of the Series *Fundamentals of Computing* is a series of four texts and accompanying laboratory manuals that provide the basis for a four-semester breadth-first introduction to the discipline of computing. This series is motivated by both the comprehensive definition of the discipline and the pedagogical principles developed in the reports *Computing as a Discipline*¹ and *Computing Curricula 1991*.² This introduction to the discipline has the following themes:

1. A broad treatment of the nine major subject areas of the discipline
2. A grounding in the mathematical, scientific, and engineering points of view on the discipline, known as theory, abstraction, and design, respectively
3. A responsible treatment of key social, ethical, and legal issues that uniquely concern the discipline and the profession
4. A scheduled weekly laboratory experience, with separate accompanying laboratory manuals and software tools
5. A carefully developed methodology for algorithmic problem solving (*MAPS*)

The major subject areas of computing are:

Algorithms and data structures
Architecture
Artificial intelligence and robotics

Database and information retrieval
 Human-computer communication
 Numerical and symbolic computation
 Operating systems
 Programming languages
 Social, ethical, and professional issues
 Software methodology and engineering

The four volumes in this *FUNDAMENTALS OF COMPUTING* series are titled

Volume I: Logic, Problem Solving, Programs, and Computers

Volume II: Abstraction, Data Structures, and Large Software Systems

Volume III: Levels of Architecture, Languages, and Applications

Volume IV: Algorithms, Concurrency, and the Limits of Computation

In addition to its comprehensive treatment of the discipline, this series provides another important dimension for the introductory courses: an option to integrate the topics of discrete mathematics with those subjects in computing where they are used. While this series does not require discrete mathematics to be integrated in all courses that use it, many instructors will choose to include that material directly in those courses. Such a choice is mainly justified on the grounds that students can understand more clearly the fundamental mathematical dimensions of the discipline than they would if these topics were taught in a separate discrete mathematics course.

A more complete discussion of the entire series and its aims can be found in the publication, "A Breadth-First Introductory Curriculum in Computing."³

Overview of this Text This text, along with its accompanying laboratory manual and software, is designed to cover an introductory course in computing, customarily known as CS1 and recently identified as C101 in the Report *Computing Curricula 1991*. These materials were first classroom-tested at Allegheny College and Bowdoin College during the 1990–1991 academic year. Following a round of revisions, they are currently in use again at Allegheny and Bowdoin, and are being classroom-tested for the first time at West Chester University and the University of Connecticut.

The content and organization of this text provide a broader view of computers, programs, problem solving, and their underlying theory than does the traditional approach to the first course. The integrated laboratory experience provides a rich experience in programming and problem solving, as in a traditional course. The laboratory component also provides hands-on experience with a von Neumann machine architecture and assembler by way of a simple simulator called MARINA.

The text has nine chapters and is organized for a one-semester course. Chapter 1 is brief and motivational in nature, concentrating on fundamental aspects of the history and nature of computing and its relationship to society. Chapters 2 and 3 cover topics of theory in the discipline—sets, functions, and an introduction to logic. These topics underlie two major areas in the discipline, software methodology and architecture.

Chapters 4 through 6 develop fundamental principles of software methodology, using preconditions and postconditions for precise specification, problem solving, and program testing and verification. MAPS, a methodology for algorithmic problem solving, is developed and illustrated. Functions and logic provide fundamental support for these activities. The principle of software reuse is also introduced here, through the notion of a routine. Techniques of proof that were introduced in Chapter 3 are reinforced here, as students see its use in the verification of simple programs. Though theory is utilized in these chapters, their main purpose is to introduce students to the processes of abstraction and design—developing and exercising computational models of algorithmic problems using the MAPS methodology.

Chapter 7 reveals the strong influence of logic in an entirely different area of the discipline of computing—architecture. The chapter introduces principles of logic design and machine organization, so that students ultimately see the interconnection between the Pascal programs that they developed earlier in the course and the computers that execute these programs. The main focus of this chapter is on the process of design as it influences the architecture and functional characteristics of contemporary computers.

Chapter 8 takes an entirely different point of view toward the discipline. It discusses two important aspects of the social context of the computing discipline—intellectual property and risks and liabilities. At this point, students are familiar enough with the ideas of software, correctness, error detection and correction, and architecture to think constructively about these fundamental contemporary issues.

Finally, Chapter 9 provides a broader overview of the remaining areas in the discipline of computing. It discusses the areas of algorithms and data structures, numerical and symbolic computation, operating systems, database and information retrieval, artificial intelligence and robotics, and human-computer communication. It also presents some of the broad professional aspects of the computing discipline. Thus, students

confront subjects that they are likely to encounter if they decide to major or minor in computer science or computer engineering.

It is important to note that this text can be used independently of Volumes II through IV. That is, the quality and range of its subject matter will provide excellent preparation for later courses in the curriculum, whether those courses have the organization of Volumes II-IV or more conventional course organizations. Finally, this text can also be used to introduce nonmajors to the discipline of computing in the setting of a service course. More discussion of these alternatives is given below.

Course Organization—The Integrated Breadth-First Path When classroom-tested in a 14-week one-semester introductory course, the topics in this text were covered as shown in the table below. We identify this particular path through the text as the *Integrated Breadth-First Path*. The weekly schedule required 3 hours of lectures and a separate 1-hour coordinated laboratory period. The laboratory itself can be equipped either with IBM PC (compatible) computers running Turbo Pascal or with Macintosh computers running THINK Pascal. Other configurations that support standard Pascal can also be used. Nevertheless, the instructor should prepare for 4 scheduled hours of student contact per week rather than 3.

Week	Topics	Text Chapters	Coordinated Homework
1	History of computing, review of functions and sets, finite series	1, 2	Exercises, chapter 2
2-3	Introduction to logic, equivalences, quantifiers, methods of reasoning and proof	3	Exercises, chapter 3
4	Introduction to algorithmic problems and their solutions; specifications	4	
5-6	Methodology for algorithmic problem solving (MAPS); routines, libraries, and reuse	5	
7-9	Using MAPS to solve text and graphics problems; correctness, testing, verification	6	Exercises, chapter 6
10-12	Logic, circuits, and computer organization; machine language and assembly language	7	Exercises, chapter 7
13-14	Social issues; software as intellectual property; overview of the discipline	8, 9	Short paper

From our experience, an ideal laboratory size would have 20 or fewer students, with at least one computer for every two students; it is often pref-

CONTENTS

FOREWORD	xv
PREFACE	xix
CHAPTER 1: COMPUTING AS A HUMAN ENTERPRISE	1
1.1 A BRIEF HISTORY OF COMPUTING	1
1.2 THEORY, ABSTRACTION, AND DESIGN	9
1.3 THE NINE SUBJECT AREAS OF COMPUTING	11
1.4 COMPUTING AND DAILY LIFE: A "COOK BOOK" EXAMPLE	12
1.5 SUMMARY	14
Exercises	15
CHAPTER 2: SETS AND FUNCTIONS	17
2.1 SETS	18
2.1.1 Set Relationships: Venn Diagrams	19
2.1.2 Variables, Types, and States	20
2.1.3 Set Operations	23
2.1.4 Properties of Set Operations	26
2.1.5 Sets of Strings	27
Exercises	29
2.2 FUNCTIONS	31
2.2.1 Basic Concepts	33
2.2.2 Continuous and Discrete Functions	34
2.2.3 Alternative Ways of Defining Functions	36
Exercises	38
2.2.4 One-to-one Functions and Inverses	40
2.2.5 Boolean, Integer, Exponential, and Logarithmic Functions	44
Exercises	48
2.2.6 Finite Series and Related Functions	49
2.3 SUMMARY	54
Exercises	54
CHAPTER 3: LOGIC	57
3.1 PROPOSITIONAL LOGIC	57
3.1.1 Representing English Statements Using Propositional Logic	60
3.1.2 Evaluating Propositions: Truth Values	63

3.1.3 Tautologies	67
Exercises	69
3.2 REASONING WITH PROPOSITIONS	70
3.2.1 Equivalence	71
3.2.2 Properties of Equivalence	72
3.2.3 Rules of Inference: The Idea of Proof	75
3.2.4 Proof Strategies	78
3.2.5 Solving Word Problems	82
Exercises	84
3.3 PREDICATE LOGIC	87
3.3.1 The Universal and Existential Quantifiers	90
3.3.2 Further Quantifiers	94
3.3.3 Free and Bound Variables	95
3.4 PREDICATES AND PROGRAMS	95
3.4.1 The State of a Computation	96
3.4.2 Quantifiers and Programming: Loops	96
Exercises	98
3.5 REASONING WITH PREDICATES: PROOF BY INDUCTION	100
3.6 SUMMARY	106
Exercises	106
 CHAPTER 4: ALGORITHMIC PROBLEMS AND THEIR SOLUTIONS	 109
4.1 ALGORITHMS AND PROBLEMS	110
Exercise	113
4.2 PROBLEM DEFINITION AND ALGORITHM DESCRIPTION	113
4.2.1 The Initial and Final States of an Algorithm: Input and Output	114
4.2.2 The Intermediate States of a Computation: Introducing Variables	115
Exercises	118
4.3 ALGORITHMIC LANGUAGE	118
4.3.1 Syntax and Semantics	121
4.3.2 Iteration and Loops: Initialization, Invariance, Termination	123
4.3.3 Three Views of the Same Problem Solution	129
Exercises	131
4.4 MORE ALGORITHMIC PROBLEMS	133
4.4.1 Computing a^b	133
4.4.2 Counting Words in a Text	137
4.4.3 Monitoring a Tic-Tac-Toe Game	142
Exercises	147
4.5 SUMMARY	150
 CHAPTER 5: SOLVING ALGORITHMIC PROBLEMS	 153
5.1 WE NEED A METHODOLOGY	153

5.1.1 Overview of the MAPS Methodology	154
5.2 DEVELOPING SOFTWARE FOR REUSE: THE ROUTINE	156
5.2.1 Encapsulating Routines for Reuse: Procedural Abstraction	156
5.2.2 Identifying New Routines and Defining Their Abstractions	162
5.2.3 Recursive Functions: An Alternative to Iteration	166
5.2.4 Learning About Existing Routines: Libraries and Language Features ...	168
5.2.5 Selection and Reuse of Data Types and Structures	170
5.2.6 Arrays of Strings	171
5.2.7 Strong Typing and Coercion	173
Exercises	175
5.3 A CASE STUDY IN PROBLEM SOLVING USING MAPS	178
5.3.1 The Dialogue	178
5.3.2 The Specifications	180
5.3.3 The Breakdown	181
Exercises	183
5.4 DEFINING ABSTRACTIONS: UNITING OLD ROUTINES WITH NEW IDEAS	184
5.4.1 Reusing Routines	185
5.4.2 Using Creativity to Design New Routines	188
5.4.3 Using Domain Knowledge to Design New Routines	189
5.5 COMPLETING THE CASE STUDY	191
5.5.1 Coding	192
5.5.2 Testing and Verification	194
5.5.3 Presentation	194
5.6 SUMMARY	196
Exercises	196
 CHAPTER 6: ALGORITHM ROBUSTNESS AND TESTING ...	199
6.1 CORRECTNESS AND ROBUSTNESS	200
Exercises	203
6.2 SOLVING TEXT PROCESSING PROBLEMS WITH MAPS: CRYPTOGRAPHY	204
6.3 SOLVING GRAPHICS PROBLEMS WITH MAPS: THE GAME OF LIFE	208
Exercise	218
6.4 ENSURING ROBUSTNESS: TEST CASE DESIGN	218
6.4.1 Example: Testing a Complete Procedure or Function	219
6.4.2 Example: Testing a Complete Program	223
6.5 ENSURING CORRECTNESS: PROGRAM VERIFICATION	225
6.5.1 The Proof Tableau	225
6.5.2 The Assignment Rule of Inference	227
6.5.3 Reusing the Rules of Inference from Logic	230
6.5.4 Rules for Conditionals	231
6.5.5 Verifying Loops	234

6.5.6 Formal Versus Informal Program Verification	237
6.6 SUMMARY	238
Exercises	239
CHAPTER 7: COMPUTER REALIZATION OF ALGORITHMS	241
7.1 OVERVIEW OF COMPUTER ORGANIZATION	242
7.1.1 Input	243
7.1.2 Storage	243
7.1.3 The Processor	243
7.1.4 Output	245
7.2 NUMBER SYSTEMS	246
Exercises	249
7.3 REPRESENTATION OF INFORMATION IN A COMPUTER	250
7.3.1 Unsigned Integers	250
7.3.2 Signed Integers	251
7.3.3 Internal Representation of Characters	254
Exercises	255
7.4 MEMORY	256
7.4.1 Memory Organization	256
7.4.2 Storage Implementation	258
7.5 THE ARITHMETIC-LOGIC UNIT	262
7.5.1 Registers	265
7.5.2 Operational Units	267
7.5.3 Designing an Adder	272
7.5.4 Subtraction	277
7.5.5 Combining the Operational Units to Make an ALU	280
Exercises	282
7.6 THE CONTROL UNIT	282
7.6.1 Programs and Instructions	283
7.6.2 The Instruction Set	287
7.6.3 Loops	289
7.6.4 Processing Instructions	292
7.6.5 Bringing It All Together	297
Exercises	299
7.7 MARINA: A SIMULATED COMPUTER	299
7.7.1 MARINA's Memory	302
7.7.2 MARINA's Registers	303
7.7.3 MARINA's Instruction Format	303
7.7.4 Assembly Language	306
7.7.5 Assembly Language Data and Variable Declarations	309
Exercises	312
7.8 SUMMARY	314