PROBLEM SOLVING and COMPUTER PROGRAMMING

Peter Grogono
Sharon H. Nelson

PROBLEM SOLVING AND COMPUTER PROGRAMMING

PETER GROGONO
Department of Computer Science, Concordia University, Montreal

SHARON H. NELSON
Metonymy Productions, Montreal

ADDISON-WESLEY PUBLISHING COMPANY
Reading, Massachusetts • Menlo Park, California
London • Amsterdam
Don Mills, Ontario • Sydney

Library of Congress Cataloging in Publication Data

Grogono, Peter.
Problem solving and computer programming.

Bibliography: p.
Includes index.
1. Electronic digital computers—Programming.
2. Mathematics—Problems, exercises, etc.—Data processing. 3. Problem solving—Data processing. I. Nelson, Sharon H., 1948- II. Title.
QA76.6.G758 519.4 81-7942
ISBN 0-201-02460-8 AACR2

ISBN 0-201-02460-8
ABCDEFGHIJ-AL-898765432

PREFACE

This book has grown out of our perception of a problem. It has become common practice to combine the subject of problem-solving with a tutorial for a particular programming language in a single text. We believe that this practice is misleading; problem solving and computer programming are distinct activities. If we are taught to associate problem-solving methods with a particular language, our thinking is likely to become dominated by that language. This is as true of programming languages as it is of natural languages. If we learn to "think in Pascal" we may fail to find solutions that are expressed easily in LISP or SNOBOL, or we may get stuck with unwieldy solutions, or our ability to find insights into problems may be severely limited.

The structure of this book, from the separation into two major parts to the placement of chapters and examples within chapters, is intended to present a thematic rather than a linear approach. Programming problems, like many other problems, are amenable to solution by general problem-solving techniques and strategies. Like other problems, programming problems can be divided into subproblems, and the subproblems can be further subdivided, until we reach a level at which solutions are known or can easily be found. In programming, the first subproblem is to discover an appropriate algorithm and the second subproblem is to implement the algorithm by writing a program. General problem-solving methods, of the kind that are developed in Part 1 of

this book, can be applied to both of these subproblems. Part 1 presents techniques and strategies which are not only aids to general problem solving but are also aids to good programming. Foundations for the programming sections of Part 2 are developed in Part 1, and the programming methodology of Part 2 is based on the tactics and strategies of Part 1.

There are many approaches to programming. For example, software engineers consider that writing a program is like building a bridge and mathematical programmers consider that writing a program is like proving a theorem. The problem-solving approach is not an alternative but is rather a complement to these approaches. It emphasizes the conceptual nature of programming, the numerous ways of obtaining insight into programming problems, and the distinction between a high-level algorithmic solution and a low-level implementation of that solution.

The discipline that has contributed most to problem solving is mathematics. This is not surprising; successful problem solving requires a high level of abstraction, the elimination of superfluous details, and appropriate notations. Mathematics is the language of abstraction and the source of many notations. Consequently, topics in discrete mathematics, such as induction, propositional calculus, and graph theory, are introduced in Part 1. These topics are not discussed in depth and supplementary reading in discrete mathematics is strongly recommended. Although technical skill in formal mathematics is not a prerequisite for programming, it is undoubtedly very useful, and for advanced programming the mathematician's concern with abstraction and precision is essential.

In Part 2 of this book we discuss the particular problem of writing a computer program. Although a large part of the work of designing a program can and should be done independently of a particular programming language, and is done in this way in this book, a language is needed if complete solutions are to be presented. The language in which examples are written is Pascal and the notation used for algorithms is based on Pascal. Pascal has been chosen because it has a simple and useful set of structuring techniques for control and data and it therefore provides an appropriate basis for an algorithmic notation. This notation, or something close to it, would be useful for writing programs in a procedural language other than Pascal. The development of each program in this book is largely independent of any particular programming language and would have been almost the same if another language had been used. (Although we have attempted to avoid using Pascal idioms, we have used the with statement to achieve compactness and we have not been able to avoid using the Pascal dereferencing operator.)

This book is intended for readers who already have some programming experience and who are familiar with Pascal or with another language with comparable features, such as PL/I or Algol-68. Programs are developed only as far as is consistent with a problem-solving approach. Complete versions of the programs would validate and echo their input data, would issue comprehensive error reports when necessary, and would solve a more general class of problems. They would also occupy a large amount of space and would contain much material irrelevant to the text. The programs contain few comments because the surrounding text provides adequate commentary. The same programs would be written differently and would contain more comments in a different environment. We must admit that we are not entirely happy with this situation, but we have yet to see a program suitable both for inclusion in a text book and for actual use.

Sometimes a discussion of a problem is separated from a related example or a program appears several chapters after the discussion it illustrates. This unconventional ordering is intended to expose underlying concepts rather than to quickly provide solutions for given problems. The point of a problem-solving approach is that it reveals strategies, techniques, and insights that, once discovered, may be applied in many different situations. Example programs are not meant to stand as particular solutions but are intended to illustrate concepts. Chapters are built around topics and sections are built around discussions so that the development of concepts rather than the evolution of particular solutions becomes the focus for study.

Most of the problems used to illustrate Part 1 are traditional. They are described in numerous anthologies, notably the excellent and enjoyable collections of Martin Gardner. These problems are not presented as a challenge to the reader, who has probably encountered them before, but rather for the light they shed on problem-solving techniques. Some of these problems are very old; one of them appeared in an Indian instructional text some 3000 years ago. There must be some reason for this longevity, and it may be the insight these problems provide into our ability to think and to solve problems. A further motivation for the inclusion of traditional problems is that they demonstrate the continuity of thought. We ought not to jettison thousands of years of intellectual work and achievement merely because modern technology has created a few new problems. We can often benefit by reexamining old problems and their solutions; such examination provides an understanding of how we reach solutions. From this understanding we can build not only new solutions for new problems but new solutions for old problems. Working through example problems will help us to develop insight into the tech-

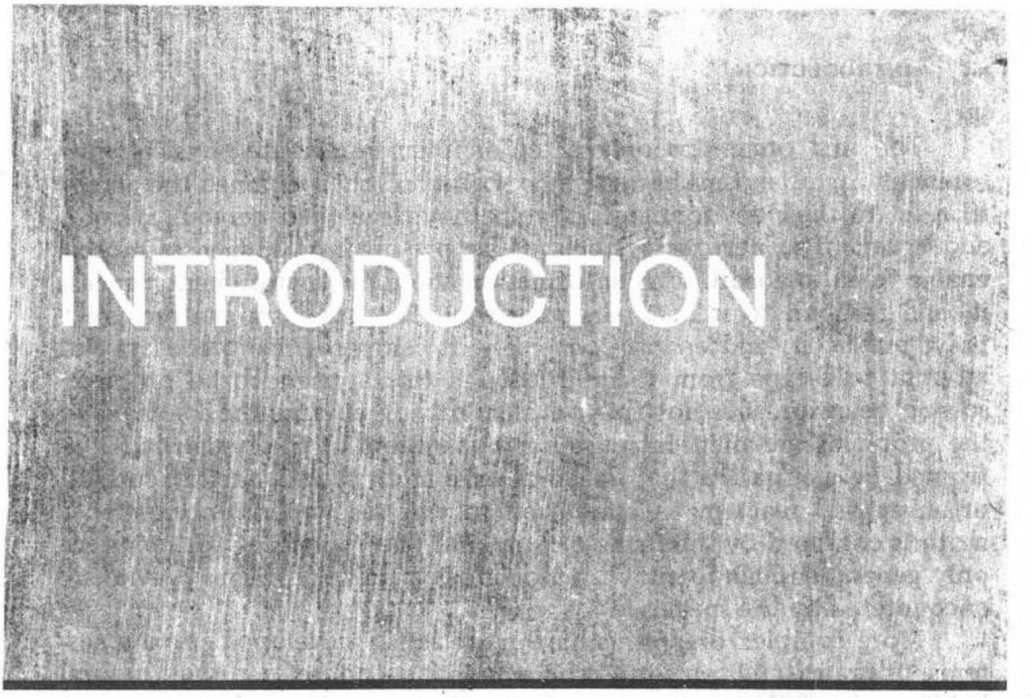
niques that each of us finds most natural and comfortable as well as insight into the examples themselves.

As we examined and worked through traditional problems, we realized that the traditional expression of many of these problems reflects embedded values and prejudices with which many of us are not comfortable. This discomfort accounts for the rewriting or recasting of some of the problems.

This book touches on many subjects. At the end of each chapter there are suggestions for further reading. In these sections, books and papers are referred to by author and title; full citations are given in the bibliography at the end of the book.

Many people have helped us during the preparation of this book. We are grateful to John Gannon, Henry Ledgard, Gary Boyd, and a number of others who reviewed various versions of the manuscript. Joel Hillel kindly invited one of us (P.G.) to attend his problem-solving seminars, and these seminars provided valuable background material for Part 1.

Montreal P.G.
March 1981 S.H.N.



Life began, as far as we can tell, when a complex mixture of chemical compounds was so arranged that the most fundamental problems of living were solved. These problems were, and are, obtaining food, reproducing, and defending the organism against predators. All species must solve these problems or become extinct.

We do not know if there ever existed an organism that could only reproduce itself exactly; we do know that the reproductive mechanisms of all existing species allow small changes to occur between one generation and the next. Thus at any given time the individuals of a species will be slightly different from one another. If the environment changes slightly, the new conditions will favor some individuals and these individuals will tend to live longer and to produce more offspring than others. In this way, the population as a whole solves the problem of adaptation to a changing environment. This is the mechanism of evolution expressed in its barest form.

Evolution is a slow process which works best for a simple organism that has a short life cycle and lives in a slowly changing environment. Complex organisms, which have longer life cycles and live in more complex environments, require the ability to solve problems as they arise and the ability to do so as individuals rather than as a species over many generations.

The first organisms only acted on their immediate physical environment. As organisms became more complex, they attained the ability to react to their environment as well: they developed nervous systems and brains. The nervous systems of simple organisms, such as moths, enable them to absorb some information about the environment, process it, and respond to it in a particular way. The organism can recognize prey, pursue it, and eat it; conversely, it can recognize predators and attempt to escape from them. Although this appears to be problem-solving behavior, it is not true problem solving because the solutions to the problems are built into the nervous system of the organism. Confronted by a situation that does not arise in its natural habitat, such an organism will react in an inappropriate way, as, for instance, when a moth is confused by the flame of a candle. The "wired-in" solutions are only general enough to allow an individual to solve problems previously encountered by the species.

More complex organisms have more highly developed nervous systems, or brains, and can solve problems that do not arise in their natural habitats. A rat, for example, can be taught to obtain food by pressing a lever or running through a maze. This is true problem solving.

All species have evolved mechanisms by which to survive. Teeth, claws, horns, camouflage, the ability to run, climb, hide, and fly are all characteristics that allow members of a species to survive and perpetuate the species. Some species have also evolved the ability to think, and for a few species this ability has become the dominant survival mechanism. We are members of the genus *homo*, species *sapiens*: we have weak teeth, feeble claws, no horns, and we can neither run fast nor fly, but, for better or for worse, we are able to think and to solve some problems.

The English word "problem" is derived from a classical Greek word, *proballein*, which meant "something thrown forward." The root of this word, *ballein*, with a different prefix, *syn*, gave us the word *symbol* which, if it were true to its etymology would mean "thrown together with." Yet another prefix, *dia*, yielded the Greek word *diaballein*, which became *diabolos* and eventually the English word "devil." We are indeed "bedeviled" by something "thrown across" our paths from time to time. A problem is *that which is forced into our awareness, a challenge*, something to be overcome or crossed over. The overcoming or crossing over are often, however, physical activities only in a metaphorical sense; we solve problems first of all not by climbing or jumping over obstacles but by the mental effort of envisioning solutions. Although we may be haunted or tormented or kept awake at night by some devilishly difficult problem, our solutions are arrived at not so

much by mysterious rites or the preparation of potions but by methodical and often logical thought processes.

As thinking beings we are constantly involved in solving problems. How do I get to the other side of the street? How can I finish my assignment by Friday? Shall I have supper for less than $5 or can I get to the bank by four o'clock? Whom shall I marry?

These are problems of the kind that the evolution of our species and our own upbringing have taught us to solve, and although they are interesting problems, they are not discussed further in this book. They are too complicated, too diverse in their ramifications, and too vaguely defined to be amenable to solution by direct application of the methods we will discuss. Nevertheless, this book may help you to solve some problems of this kind, or at least give you some insight into how rational thinking might contribute to their solution.

The problems we encounter during the course of our work are usually related to the work we do; the problems a doctor is expected to solve concern sick people and the problems a plumber is expected to deal with concern malfunctioning pipes. Although people outside these two disciplines may detect a humorous relationship between them, the tools and knowledge that each practitioner brings to the work are quite different.

In a study of general problem-solving techniques we do not want to introduce specialized knowledge of medicine, plumbing, or other fields. Accordingly, the problems discussed in Part 1 of this book are puzzles and brain teasers. Most of these problems will probably be familiar to you and you will perhaps know the solutions to many of them. This will not detract from their function, which is to illustrate problem-solving methods.

The concept of mechanical computation is not new. The slide rule was invented in the seventeenth century by Delamain and Oughtred, both of whom taught mathematics in London. Pascal built the first mechanical calculating machine in 1642; the principles of this machine were used until electronic calculators replaced mechanical calculators a few years ago. Carillons, music boxes, and automatic looms all have a long history. What *is* new is the existence of a technology that enables extremely complex programs to be executed rapidly and precisely. This technology, in the form of electronic computers, has given rise to a craft called *programming* and to an assortment of craftspersons called *programmers*.

In Part 2 of this book we adopt the point of view that in writing a computer program we are solving a problem. The problem is always of the form, "make the machine do so-and-so" and the solution is a pro-

gram that causes "so-and-so" to happen. This problem is amenable to solution by any or all of the techniques discussed in Part 1.

There are many approaches to programming other than through problem solving. Most of these approaches stress the importance of quality in the final product. At a time when computer systems are used to implement financial transactions, control civilian aircraft, design buildings and bridges, and guide satellites to distant planets, we cannot afford to underestimate the importance of software quality. The emphasis on the quality of the final product may, however, lead to the neglect of the early phases of program construction. Early attention to program specification and design can aid us greatly in the production of high quality software. How do we choose appropriate algorithms, or if there are no appropriate algorithms, how do we invent the algorithms we need? It is at this stage that problem-solving techniques are most useful. We need means of obtaining insight, of decomposing complex problems into simpler subproblems, and notations in which to express our solutions.

PART 1

PROBLEM SOLVING

CONTENTS

原书缺页

原书缺页