Programming in Haskell

Second Edition

Haskell is a purely functional language that allows programmers to rapidly develop clear, concise and correct software. The language has grown in popularity in recent years, both in teaching and in industry. This book is based on the author's experience of teaching Haskell for more than 20 years. All concepts are explained from first principles and no programming experience is required, making this book accessible to a broad spectrum of readers. While Part I focuses on basic concepts, Part II introduces the reader to more advanced topics.

This new edition has been extensively updated and expanded to include recent and more advanced features of Haskell, new examples and exercises, selected solutions, and freely downloadable lecture slides and code. The presentation is clean and simple, while also being fully compliant with the latest version of the language, including recent changes concerning applicative, monadic, foldable and traversable types.

Graham Hutton is Professor of Computer Science at the University of Nottingham. He has taught Haskell to thousands of students and received numerous best lecturer awards. Hutton has served as an editor of the *Journal of Functional Programming*, chair of the Haskell Symposium and the International Conference on Functional Programming, vice-chair of the ACM Special Interest Group on Programming Languages, and he is an ACM Distinguished Scientist.

Programming in Haskell

Second Edition

GRAHAM HUTTON
University of Nottingham

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
4843/24, 2nd Floor, Ansari Road, Daryaganj, Delhi - 110002, India
79 Anson Road, #06-04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781316626221
10.1017/9781316784099

First published 2007
Second edition 2016

Printed in the United Kingdom by Clays, St Ives plc in 2016

A catalogue record for this publication is available from the British Library

ISBN 978-1-316-62622-1 Paperback

For Annette, Callum and Tom

Contents

Foreword

It is nearly a century ago that Alonzo Church introduced the lambda calculus, and over half a century ago that John McCarthy introduced Lisp, the world's second oldest programming language and the first functional language based on the lambda calculus. By now, every major programming language including JavaScript, C++, Swift, Python, PHP, Visual Basic, Java, ... has support for lambda expressions or anonymous higher-order functions.

As with any idea that becomes mainstream, inevitably the underlying foundations and principles get watered down or forgotten. Lisp allowed mutation, yet today many confuse functions as first-class citizens with immutability. At the same time, other effects such as exceptions, reflection, communication with the outside world, and concurrency go unmentioned. Adding recursion in the form of feedback-loops to pure combinational circuits lets us implement mutable state via flip-flops. Similarly, using one effect such as concurrency or input/output we can simulate other effects such as mutability. John Hughes famously stated in his classic paper *Why Functional Programming Matters* that we cannot make a language more powerful by eliminating features. To that, we add that often we cannot even make a language less powerful by removing features. In this book, Graham demonstrates convincingly that the true value of functional programming lies in leveraging first-class functions to achieve compositionality and equational reasoning. Or in Graham's own words, "functional programming can be viewed as a style of programming in which the basic method of computation is the application of functions to arguments". These functions do not necessarily have to be pure or statically typed in order to realise the simplicity, elegance, and conciseness of expression that we get from the functional style.

While you can code like a functional hacker in a plethora of languages, a semantically pure and lazy, and syntactically lean and terse language such as Haskell is still the best way to learn how to think like a fundamentalist. Based upon decades of teaching experience, and backed by an impressive stream of research papers, in this book Graham gently guides us through the whole gambit of key functional programming concepts such as higher-order functions, recursion, list comprehensions, algebraic datatypes and pattern matching. The book does not shy away from more advanced concepts. If you are still confused by the n-th blog post that attempts to explain monads, you are in the right place. Gently starting with the IO monad, Graham progresses from functors to applicatives using many concrete examples. By the time he arrives at monads, every reader will feel that they themselves could have come up with the concept of a monad as a generic pattern for composing functions with effects. The chapter on monadic parsers brings everything together in a compelling use-case of parsing arithmetic expressions in the implementation of a simple calculator.

This new edition not only adds many more concrete examples of concepts introduced throughout the book, it also introduces the novel Haskell concepts of foldable and traversable types. Readers familiar with object-oriented languages routinely use iterables and visitors to enumerate over all values in a container, or respectively to traverse complex data structures. Haskell's higher-kinded type classes allow for a very concise and abstract treatment of these concepts by means of the Foldable and Traversable classes. Last but not least, the final chapters of the book give an in-depth overview of lazy evaluation and equational reasoning to prove and derive programs. The capstone chapter on calculating compilers especially appeals to me because it touches a topic that has had my keen interest for many decades, ever since my own PhD thesis on the same topic.

While there are plenty of alternative textbooks on Haskell in particular and functional programming in general, Graham's book is unique amongst all of these in that it uses Haskell simply as a tool for thought, and never attempts to sell Haskell or functional programming as a silver bullet that magically solves all programming problems. It focuses on elegant and concise expression of intent and thus makes a strong case of how pure and lazy functional programming is an intelligible medium for efficiently reasoning about algorithms at a high level of abstraction. The skills you acquire by studying this book will make you a much better programmer no matter what language you use to actually program in. In the past decade, using the first edition of this book I have taught many tens of thousands of students how to juggle with code. With this new edition, I am looking forward to extending this streak for at least another 10 years.

Erik Meijer

Preface

What is this book?

Haskell is a purely functional language that allows programmers to rapidly develop software that is clear, concise and correct. The book is aimed at a broad spectrum of readers who are interested in learning the language, including professional programmers, university students and high-school students. However, no programming experience is required or assumed, and all concepts are explained from first principles with the aid of carefully chosen examples and exercises. Most of the material in the book should be accessible to anyone over the age of around sixteen with a reasonable aptitude for scientific ideas.

How is it structured?

The book is divided into two parts. Part I introduces the basic concepts of pure programming in Haskell and is structured around the core features of the language, such as types, functions, list comprehensions, recursion and higher-order functions. Part II covers impure programming and a range of more advanced topics, such as monads, parsing, foldable types, lazy evaluation and reasoning about programs. The book contains many extended programming examples, and each chapter includes suggestions for further reading and a series of exercises. The appendices provide solutions to selected exercises, and a summary of some of the most commonly used definitions from the Haskell standard prelude.

What is its approach?

The book aims to teach the key concepts of Haskell in a clean and simple manner. As this is a textbook rather than a reference manual we do not attempt to cover all aspects of the language and its libraries, and we sometimes choose to define functions from first principles rather than using library functions. As the book progresses the level of generality that is used is gradually increased. For example, in the beginning most of the functions that are used are specialised to simple types, and later on we see how many functions can be generalised to larger classes of types by exploiting particular features of Haskell.

How should it be read?

The basic material in part I can potentially be worked through fairly quickly, particularly for those with some prior programming experience, but additional time and effort may be required to absorb some of material in part II. Readers are recommended to work through all the material in part I, and then select appropriate material from part II depending on their own interests. It is vital to write Haskell code for yourself as you go along, as you can't learn to program just by reading. Try out the examples from each chapter as you proceed, and solve the exercises for each chapter before checking the solutions.

What's new in this edition?

The book is an extensively revised and expanded version of the first edition. It has been extended with new chapters that cover more advanced aspects of Haskell, new examples and exercises to further reinforce the concepts being introduced, and solutions to selected exercises. The remaining material has been completely reworked in response to changes in the language and feedback from readers. The new edition uses the Glasgow Haskell Compiler (GHC), and is fully compatible with the latest version of the language, including recent changes concerning applicative, monadic, foldable and traversable types.

How can it be used for teaching?

An introductory course might cover all of part I and a few selected topics from part II; my first-year course covers chapters 1–9, 10 and 15. An advanced course might start with a refresher of part I, and cover a selection of more advanced topics from part II; my second-year course focuses on chapters 12 and 16, and is taught interactively on the board. The website for the book provides a range of supporting materials, including PowerPoint slides and Haskell code for the extended examples. Instructors can obtain a large collection of exams and solutions based on material in the book from `solutions@cambridge.org`.

Acknowledgements

I am grateful to the University of Nottingham for providing a sabbatical to produce this new edition; Thorsten Altenkirch, Venanzio Capretta, Henrik Nilsson and other members of the FP lab for our many enjoyable discussions; Iván Pérez Domínguez for useful comments on a number of chapters; the students and tutors on all of my Haskell courses for their feedback; Clare Dennison, David Tranah and Abigail Walkington at CUP for their editorial work; the GHC team for producing such a great compiler; and finally, Catherine and Ian Hutton for getting me started in computing all those years ago.

Many thanks also to Ki Yung Ahn, Bob Davison, Philip Hölzenspies and Neil Mitchell for providing detailed comments on the first edition, and to the following for pointing our errors and typos: Paul Brown, Sergio Queiroz de Medeiros, David Duke, Robert Fabian, Ben Fleis, Robert Furber, Andrew Kish, Tomoyas Kobayashi, Florian Larysch, Carlos Oroz, Douglas Philips, Bruce Turner, Gregor Ulm, Marco Valtorta and Kazu Yamamoto. All of these comments have been taken into account when preparing the new edition.

Graham Hutton