Exploring Languages with Interpreters and Functional Programming Chapter 0

H. Conrad Cunningham

11 April 2022

Contents

Professor of Computer and Information Science
University of Mississippi
214 Weir Hall
P.O. Box 1848
University, MS 38677
(662) 915-7396 (dept. office)

Browser Advisory: The HTML version of this textbook requires a browser that supports the display of MathML. A good choice as of April 2022 is a recent version of Firefox from Mozilla.

0 Preface

0.1 Dedication

0.1.1 July 2018

I dedicate this textbook to my parents—my mother, Mary Cunningham, and my father, the late Harold "Sonny" Cunningham—and to my wife, Diana Cunningham.

I thank Mother and Dad for their love and encouragement throughout my nearly 64 years on this earth. They taught me the importance of hard work, both physical and mental. They taught me the importance of faith in God and of personal integrity. I hope I have been a good student.

I write this sitting at a desk in my mother's home late one evening in July 2018. I remember a time more than a half century ago when I was struggling with an elementary school writing assignment. Mother wrote an example page that I remember as amazing. I thank her for that encouragement. I still suffer from a deficit of creativity at times, but I was able to write this approximately 400-page textbook.

I look around and see a plaque for an award my father received for serving his church as a Sunday School teacher for 40 years. It reminds me of the many positive contributions he made to his community and his church, many unseen by others. I hope I am also making positive contributions to the various communities, physical and virtual, in which I live and work.

I thank Diana, my wife of 42 years, for her love and friendship—for being my companion on the journey of life. This textbook is an effort that has spanned more than a quarter century. She has lived it nearly as much as I have. Many times she has urged me to stop work and get some sleep, as she did just now.

0.1.2 November 2019 Addendum

My mother passed away in June 2019 at the age of 91 years and 8 months. We miss her dearly! Her family and friends will remember for as long as we live.

We love you, Mom, and look forward to the reunion of our family in heaven.

0.2 Course 1 and Course 2

As the title suggests, I designed this textbook to be used for at least two different kinds of courses:

1. A course on "functional programming" targeted at advanced undergraduate and beginning graduate students who have previously programmed using imperative languages but who have not used functional or relational languages extensively.

This *functional and modular programming* course focuses on parts of Chapter 2 and 80 and Chapters 4-30.

I have been teaching such an elective course at the University of Mississippi since 1991 (CSci 555, Functional Programming). I have been teaching the Haskell programming language since 1993. Some of the content of this textbook evolved from class notes I originally developed for the course in the 1991-6 period.

My approach to the course was initially motivated by the first edition of the classic Bird and Wadler textbook [3,4].

2. A course on programming language organization targeted at a similar audience.

   There are several approaches to teaching the programming languages course. My approach in this textbook focuses on "exploring languages with interpreters". It seeks to guide students to learn how programming languages work by developing interpreters for simple languages.

   This programming language organization course focuses on Chapters 1-3, Chapters 40-49, and parts of Chapters 4-30 as needed.

   Kamin's excellent textbook *Programming Languages: An Interpreter-Based Approach* [11] motivated my approach. But, instead of using Pascal or C to develop the interpreters as Kamin's book did, this textbook primarily uses Haskell. Other influences on my approach are the book by Sestoft [19,20], the online books by Krishnamurthi [13,14], and an early manuscript by Ramsey [16] (which is based on Kamin's book).

   I began experimenting with this approach using the Lua language in my graduate Software Language Engineering (CSci 658) course in Fall 2013. I first taught the interpreter approach (using Lua) in the required undergraduate Organization of Programming Languages (CSci 450) course at the University of Mississippi in 2016. I used Haskell with the interpreter-based approach in 2017 and 2018.

Of course, students must become familiar with basic functional programming and Haskell for Course 2 to be possible.

Most real courses will likely be a mix of the two approaches.

0.3 Motivation: "Functional Programming"

Course type 1 is a course on functional and modular programming.

As a course on *programming*, Course 1 emphasizes the analysis and solution of problems, the development of correct and efficient algorithms and data structures that embody the solutions, and the expression of the algorithms and data structures in a form suitable for processing by a computer. The focus is more on the human thought processes than on the computer execution processes.

As a course on *functional* programming, Course 1 approaches programming as the construction of definitions for (mathematical) functions and (immutable) data structures. Functional programs consist of *expressions* that use these definitions. The execution of a functional program entails the evaluation of the expressions making up the program. Thus this course's focus is on problem solving techniques, algorithms, data structures, and programming notations appropriate for the functional approach.

As a course on *modular* programming, Course 1 approaches the construction of large programs as sets of modules that collaborate to solve a problem. Each module is a coherent collection of function and data type definitions. A module hides its private features, allowing their use only within the module, and exposes its public features, enabling their use by the other modules in the program.

Course 1 is not a course on functional or modular programming *languages*. In particular, it does not undertake an in-depth study of the techniques for implementing such languages on computers. (That is partly covered in Course 2.) The focus is on the concepts for programming, not on the internal details of the technological artifact that executes the programs.

Of course, we want to be able to execute our programs on a computer and, moreover, to execute them efficiently. Thus we must become familiar with some concrete programming language and use an implementation of that language to execute our programs. To be able to analyze program efficiency, we must also become familiar with the basic techniques that are used to evaluate expressions.

The academic community has long been interested in functional programming. In recent years, the practitioner community has also become interested in functional programming techniques and language features. There is growing use of languages that are either primarily functional or have significant functional subsets—such as Haskell, OCaml, Scala, Clojure, F#, Erlang, and Elixir. Most mainstream languages have been extended with new functional programming features and libraries—for example, Java, C#, Python, JavaScript, and Swift. Other interesting research languages such as Elm and Idris are also generating considerable interest.

In this textbook, we use the Haskell 2010 language. Haskell is a "lazy" functional language whose development began in the late 1980's. We also use a set of programming tools based on GHC, the Glasgow Haskell Compiler. GHC is distributed in a "batteries included" bundle called the the Haskell Platform. (That is, it bundles GHC with commonly used libraries and tools.)

Most of the concepts, techniques, and skills learned in this Haskell-based course can be applied in other functional and multiparadigm languages and libraries.

More importantly, any time we learn new approaches to problem solving and programming, we become better programmers in whatever language we are working. A course on functional programming provides a novel, interesting, and, probably at times, frustrating opportunity to learn more about the nature of the programming task.

Enjoy the "functional programming" aspects of the course and textbook!

0.4 Motivation: "Exploring Languages with Interpreters"

Course type 2 is a course on programming language organization that emphasizes design and implementation of a sequence of interpreters for simple languages.

When we first approach a new programming language, we typically think about the *syntax* of the language—how the external (e.g., textual) representation of the language is structured.

Syntax is important, but the *semantics* is more important. The semantics defines what the language means: how it "behaves" at "runtime".

In Course 2 we primarily focus on the semantics. We express the essential aspects of a expression's structure (i.e., syntax) with an *abstract syntax tree (AST)* and then process the AST to obtain a result. For example, we may have:

- an *interpreter* that takes an AST and *evaluates* it in some environment to obtain its value
- a *transformer* that takes the AST and produces a related but different (e.g., more efficient) program in the same language
- a *compiler* that takes an AST and produces a related program in a different language

By "exploring languages with interpreters", we can better understand the semantics of the programming languages. We can learn to use languages more effectively. We can explore alternative designs and implementations for languages.

This textbook uses functional and modular programming in Haskell—a paradigm and language that most students in Course 2 do not know—to implement the interpreters. Students learn new language concepts by both learning Haskell and by building language processors.

0.5 Textbook Prerequisites

This textbook assumes the reader has basic knowledge and skills in programming, algorithms, and data structures at least at the level of a three-semester introductory computer science sequence. It assumes that the reader has programming experience using a language such as Java, C++, Python, or C#; it does not assume any previous experience in functional programming. (For example, successful completion of at least CSci 211, Computer Science III, at the University of Mississippi should be sufficient.)

This textbook also assumes the reader has basic knowledge and understanding of introductory computer architecture from a programmer's perspective. (For example, successful completion of at least CSci 223, Computer Organization and Assembly Language, at the University of Mississippi should be sufficient.)

In addition, this course assumes the reader has basic knowledge and skills in mathematics at the level of a college-level course in discrete mathematical structures for computer science students. (For example, concurrent enrollment in Math 301, Discrete Mathematics, at the University of Mississippi should suffice.) The "Review of Relevant Mathematics" chapter (appendix) reviews some of the concepts, terminology, and notation used in this course.

0.6 Author's Perspective

In the 1974 Turing Award Lecture *Computer Programming as an Art*, Donald Knuth said [12]:

> The chief goal of my work as an educator and author is to help people learn to write *beautiful programs*.

In my writing and my teaching, I hope I can emulate Knuth.

I approach writing this textbook, most of my teaching and research for that matter, from the following (opinionated) perspectives:

- The essence of computing science is programming. Programming is fun!

  Of course, by "programming" I do not mean merely "coding"—and definitely not "hacking".

  I view programming as the process: of determining what the problem is, whether a solution is needed, what the desired nature of a solution is, and whether such a solution is feasible, ethical, and socially useful; of devising specific abstractions, algorithms, and information structures that correctly, elegantly, and efficiently solve the problem; of implementing the solution effectively within the concrete resources available and validating that it indeed solves the problem; and of evolving the solution and its implementation to handle changing needs. This, of course, encompasses most of what is traditionally called computer science and software engineering.

- Abstraction is the primary tool we have to deal with the complexity we must face as programmers.

  To become good programmers, we need to learn to develop good abstractions. To learn to develop good abstractions, most of us need to work upward from lots of concrete examples and experiences.

  Perhaps instead of computer science our field should be called abstraction engineering.

- Our programs should be elegant—both conceptually in terms of their design (architecture, algorithms, data structures, use of appropriate abstraction) and physically in terms of their style (use of language features, layout, use of names, appropriate comments).

- We should rigorously describe what a program must do. For example, we can define a rigorous contract that specifies what the clients of a program must do and what the program must do in response.
- We should construct larger programs as sets of collaborating modules. The modules should be designed and constructed according to the information-hiding and abstract interface principles.
- We should design our programs to be testable and test them thoroughly.
- We should reflect upon what we have done. What about our successes and failures can we observe and exploit in the future? Did our specific problem reveal or reinforce a general principle? What can we do better next time?
- To learn a programming paradigm and language well, we should immerse ourselves in the paradigm and language for a period ot time. We need to learn to think in that paradigm and language even if it is quite different from our previous experiences.
- Although we learn to program in a particular language and paradigm, we should seek to compare how the new concepts, features, and patterns of thought apply to other approaches to programming that we have learned in the past or will in the future.
- Many tasks can be viewed as language design or language processing tasks. Language design and processing are fun!
- As much as feasible, we should make instructional materials accessible (e.g., compatible with screen readers) and available in multiple formats at a low cost.

  Toward that end, I have developed most of my new instructional materials using Pandoc's dialect of Markdown and tools compatible with Pandoc.

0.7 The Journey

Although I only began to write this textbook in Summer 2016, it is a result of a journey I began long ago. Many other writers, colleagues, students, and friends have helped me during this journey.

0.7.1 Notes on Functional Programming with Haskell

I created the course CSci 555, Functional Programming, at the University of Mississippi and first taught it during the Spring 1991 semester.

I adopted the first edition of Bird and Wadler [4] as the initial textbook for the course. I thank Richard Bird and Philip Wadler for writing this excellent textbook. I thank Jeremy Gibbons for suggesting that book in a response to an inquiry I posted to a Usenet newsgroup in Summer 1990.

I also used Wentworth's RUFL (Rhodes University Functional Language) interpreter and his tutorial [24] in the first two offerings of the course. I thank Peter Wentworth for sending me (unsolicited, in response to my Usenet post) his interpreter and tutorial on a floppy disk through snail mail from the then-sanctioned South Africa.

My approach was also shaped by my research on formal methods and my previous teaching on that topic. I created the course Program Semantics and Derivation (CSci 550) and first taught it in Spring 1990 [7,8]. I followed that with the course Theory of Concurrent Programming (Engr 664), which I first taught in Fall 1990. I thank my dissertation advisor Gruia-Catalin Roman for developing my interests in formal methods, Jan Tijmen Udding for teaching a graduate course on program derivation that piqued my interests, and the other researchers or authors who have influenced my thinking: Edsger Dijkstra, Tony Hoare, David Gries, Mani Chandy, Jayadev Misra, Edward Cohen, and many others.

For the third offering of CSci 555 in Fall 1993, I switched the course to use the Gofer interpreter for the Haskell language. I thank the international committee of researchers, including Simon Peyton Jones, Paul Hudak, Philip Wadler, and others, who have developed and sustained Haskell since the late 1980s. I also thank Mark Jones for developing the lightweight Gofer interpreter and making it and its successor HUGS widely available.

Because of the need for a tutorial like Wentworth's and an unexpected delay in getting copies of the Bird and Wadler textbook [4] from Prentice Hall that semester, I began writing, on an emergency basis, what evolved into my *Notes on Functional Programming with Haskell* [9].

Some parts of the *Notes* were based on my handwritten class notes from the the 1991 and 1992 offerings of the course. Many pages of the *Notes* were written "just-in-time" in late-night sessions before I taught them the next day. I thank Prentice Hall (now Pearson) for its delay in shipping books across the "big pond", my wife Diana Cunningham for tolerating my disruptive schedule, and my Fall 1993 students for not complaining too vigorously about a quite raw set of class notes.

I continued to develop the *Notes* for the Fall 1994 and Fall 1995 offerings of the course. In early 1996, I created a relatively stable version of the *Notes* that I continued to use in subsequent offerings of CSci 555. I thank my students and others who pointed out typos and suggested improvements during the 1993-1996 period.

I thank David Gries for encouraging me to expand these notes into a textbook. I am doing that, albeit over 20 years later than Gries intended.

I formatted the *Notes* using LaTeX augmented by BibTeX for the bibliography and makeIndex for the index. I thank Donald Knuth, Leslie Lamport, and the many others who have developed and maintained TeX, LaTeX, and the other tools and packages over four decades. They form an excellent system for creating beautiful scientific documents.

I used GNU Emacs for writing and editing the source files for the *Notes*. I thank Richard Stallman and the many others who developed, maintained, and popularized Emacs over more than four decades.

For the Spring 1997 offering of CSci 555, I started using the new HUGS interpreter and the 1st edition of Thompson's textbook [21] (now it its 3rd edition [23]). I thank Simon Thompson for writing his excellent, comprehensive introductory textbook on Haskell programming.

Over the next 17 years, I corrected a few errors but otherwise the *Notes* were stable. However, I did create supplementary notes for CSci 555 and related courses. These drew on the works of Abelson and Sussman [1], Thompson [21–23], Parnas [5,15], and others. I formated these notes with HTML, Microsoft Word and Powerpoint, or plain text.

I decided to use Haskell as one of the languages in the Fall 2014 offering of Organization of Programming Languages (CSci 450). But I needed to change the language usage from the Haskell 98 standard and HUGS to the new Haskell 2010 standard and the Glasgow Haskell Compiler (GHC) and its interactive user interface GHCi. I edited the *Notes* through chapter 10 on Problem Solving to reflect the changes in Haskell 2010.

0.7.2 Organization of Programming Languages

ACM Curriculum '78 [2] influenced how most computer science academic programs were structured when they are established in the 1970s and 1980s. It defined eight core courses, most of which are still prominent in contemporary computer science curricula.

Organization of Programming Languages (CS 8) is one of those core courses. Curriculum '78 describes CS 8 as "an applied course in programming language constructs emphasizing the run-time behavior of programs" and providing "appropriate background for advanced level courses involving formal and theoretical aspects of programming languages and/or the compilation process" [2].

I first taught the required Organization of Programming Languages (CSci 450) course at the University of Mississippi in Fall 1995. I took over that class for another instructor and used the textbook the Department Chair had already selected. The textbook was the 2nd Edition of Sebesta's book [18].

Although the Sebesta book, now in its 11th edition, is probably one of the better and more popular books for CS 8-type courses, I found it difficult for me to use that semester. It and its primary competitors seem to be large, expensive tomes that try to be all things to all instructors and students. I personally find the kind of survey course these books support to be a disjointed hodgepodge. There is much more material than I can cover well in one semester. I abandoned the Sebesta book mid-way through the semester and have never wanted to use it again.

I had a copy of Kamin's textbook [11] and used two of its interpreters after abandoning Sebesta's book. It seemed to work better than Sebesta. So I ended the semester with a positive view of the Kamin approach.

My only involvement with CSci 450 for the next 18 years was to schedule and staff the course (as Department Chair 2001-15). In 2013, I began planning to teach CSci 450 again.

I decided to try an experiment. I planned to use the Kamin approach for part of the course but to redevelop the interpreters in the Lua language. Lua is a minimalist, dynamically typed language that can support multiple paradigms.

I chose Lua because (a) learning it would be a new experience for almost all of my students, (b) as a small language it should be easy for students to learn, (c) its flexibility would enable me to explore how to extend the language itself to provide new features, (d) its associated LPEG library supported the development of simple parsers, and (e) its use in computer games might make it interesting to students. I thank the Lua authors—Roberto Ierusalimschy, Waldemar Celes, and Luiz Henrique de Figueiredo—for developing this interesting platform and making it available. I thank Ierusalimschy for developing LPEG and for writing an excellent textbook on Lua programming [10].

I used the Fall 2013 offering of my Software Language Engineering (CSci 658) course to explore Lua programming and interpreters. I thank the students in that class for their feedback and suggestions—Michael Macias, Blake Adams, Cornelius Hughes, Zhendong Zhao, Joey Carlisle, and others.

However, in Summer 2014, I did not believe I was ready to undertake the interpreter-based approach in the large Fall 2014 class. Instead, I planned to try a multiple paradigm survey. I planned to begin with Haskell statically typed functional programming (using my *Notes*), then cover Lua dynamically typed, multiparadigm programming, and then use the logic language Prolog. I had taught Haskell, Lua, and Prolog in elective courses in the past.

I was comfortable with the Haskell part, but I found a required course a more challenging environment in which to teach Haskell than an elective. Covering Haskell took nearly two-thirds of the semester, leaving Lua in one-third, and squeezing out coverage of the logic language and most of the interpreter material.

I was scheduled to teach CSci 450 again in Fall 2016. For this offering, I decided to (a) begin with Lua and then follow with Haskell (the reverse order from 2014) and (b) to use the interpreter approach in the Lua segment. I adopted the 4th edition of Scott's textbook [17] to support the general material (but did not use the book much).

Unfortunately, that offering suffered from immature teaching materials for both

Lua and for the interpreter approach. I was unable to invest sufficient time in Summer 2016 to prepare course materials and revise the interpreters. Also, students, who mostly had experience with Java, had considerable difficulty modifying and debugging the dynamically typed Lua programs with 1000+ lines of code. (For various reasons, I decided to use the new Elm language instead of Haskell in the last three weeks of the semester.)

I thank the students in the Fall 2014 and Fall 2016 CSci 450 classes for giving me valuable feedback on what works and what does not—much more on the latter than the former. I also thank my Teaching Assistant (TA) for CSci 450 in the Fall 20a6 semester, Ajay Sharma, for his assistance. I learned that a large, required course like CSci 450 needs more mature teaching materials and tools than a small, elective course does. It should have been obvious!

0.7.3 Exploring Languages with Interpreters and Functional Programming

In Summer 2016, I participated in the eLearning Training Course (eTC) at the University of Mississippi to become eligible to teach online. As a part of that course, I was expected to prepare a syllabus and at least one module for some class. I chose to focus on CSci 555, Functional Programming.

This stimulated me to begin reorganizing my previous *Notes on Functional Programming with Haskell* to be a textbook for an online course on functional programming. I thank the eTC instructors Patty O'Sullivan and Wan Latartara, for (unintentionally) pushing me to begin developing this textbook.

For the textbook, I expanded the *Notes* by adapting materials I had originally developed for other purposes—such as papers with former graduate students Pallavi (Tadepalli) Darbhamulla, Yi Liu, and Cuihua Zhang—and some notes from my courses on functional programming, multiparadigm programming, software architecture, and software language engineering. I thank Darbhamulla, Liu, and Zhang. I also thank former graduate student James Church (author of a Haskell-based book [6]) for his feedback and encouragement to repackage my class notes as a textbook.

Unfortunately, I devoted too much time to this project in Summer 2016 and not enough to developing Lua-based materials and tools for the Fall 2016 offering of CSci 450, as I discussed above.

The eTC also sensitized me to the need to produce accessible instructional materials (e.g., materials compatible with screen readers for the visually impaired). I decided to expand my use of Pandoc-flavored Markdown and the Pandoc tools for producing materials in a variety of accessible formats (HTML, possibly LaTeX/PDF).

In Summer 2016, I had materials in a variety of formats. The *Notes on Functional Programming with Haskell* used LaTeX, BibTeX, and makeIndex. This is a great format for producing printed scientific documents, but not as good for display on the Web. Some of my other materials used HTML, which is great for the Web, but not for printed documents. I also had some material in Microsoft Office formats, Pandoc-flavored Markdown, and plain text (e.g., program comments).

Pandoc-flavored Markdown offered a means for achieving both greater flexibility and greater accessibility. Of course, sometimes I have to compromise on the appearance in some formats.

The Pandoc tool uses a language-processing approach, is implemented in Haskell, and supports Lua as its builtin scripting language. So it is a good fit for this textbook project. I thank John MacFarlane and many others who have developed and maintained the excellent Pandoc tools.

In Spring and Summer 2017, I combined the efforts from the previous years and sought to expand the Haskell-based functional programming course materials to include materials for the interpreter-based approach to the programming languages course and new Haskell-related material on type classes.

I redirected the work from developing materials for an online course to developing a textbook for the types of courses I describe in the "Course 1 and Course 2" section above.

In Fall 2017, I taught CSci 450 from the 2017 version of the textbook. Given my more mature materials, it worked better than the Lua-based course the previous year. But that effort identified the need for additional work on the textbook: shorter, more focused chapters, some explicit discussion of software testing, more attention to the language-processing issues, etc.

I thank my Fall 2017 and Fall 2018 TA for CSci 450, Kyle Moore, for his suggestions and corrections in a number of areas. I also thank the Spring 2017 Multiparadigm Programming (CSci 556) and Fall 2017 CSci 450 students for their feedback on the textbook materials.

I thank my long-time colleague, and current Department Chair, Dawn Wilkins, for her general suggestions on the CSci 450 course and the textbook and for the Department's continued support of my textbook writing efforts.

I also thank Armando Suarez, my Spring 2018 TA for Senior Project and student in Software Language Engineering that semester, for his suggestions on my materials and approach to those courses—some of which I have applied to this textbook and its associated courses.

In 2018, I began restructuring the 2017 version of the textbook to better meet the needs of the CSci 450 course. I changed the title to *Exploring Languages with Interpreters and Functional Programming.*

I incorporated additional chapters from the *Notes on Functional Programming with Haskell* and other materials that had not been previously included. I also developed new chapters on software testing, the language processing pipeline, and the Imperative Core language interpreter. I plan to develop additional interpreters, such as one for a Scheme-like language.

Because of this project, I have come to appreciate how much time, effort, and attention to detail must be invested to develop a good programming language organization textbook. I think Samuel Kamin, Robert Sebesta, Michael Scott, Norman Ramsey, Shriram Krishnamurthi, and other authors for their investment in writing and updating their books.

I retired from the full-time faculty in May 2019. As one of my post-retirement projects, I plan to continue work on this textbook. However, the work went slowly until 2022 because of the COVID-19 pandemic disruptions, my continued work with two PhD students until mid-2021, and various personal factors. In January 2022, I began refining the existing content, integrating separately developed materials, reformatting the document (e.g., using CSS), constructing a bibliography (e.g., using citeproc), and improving the build workflow and use of Pandoc features.

As I have noted above, I maintain this preface as text in Pandoc's dialect of Markdown using embedded LaTeX markup for the mathematical formulas and then translate the document to HTML, PDF, and other forms as needed. I continue to learn how better to apply the Pandoc-related tools to accomplish this.

0.8 References

[1] Harold Abelson and Gerald Jockay Sussman. 1996. *Structure and interpretation of computer programs (SICP)* (Second ed.). MIT Press, Cambridge, Massachusetts, USA. Retrieved from https://mitpress.mit.edu/sicp/

[2] Richard H. Austing, Bruce H. Barnes, Della T. Bonnette, Gerald L. Engel, and Gordon Stokes. 1979. Curriculum '78: Recommendations for the undergraduate program in computer science: A report of the ACM curriculum committee on computer science. *Communications of the ACM* 22, 3 (1979).

[3] Richard Bird. 1998. *Introduction to functional programming using Haskell* (Second ed.). Prentice Hall, Englewood Cliffs, New Jersey, USA.

[4] Richard Bird and Philip Wadler. 1988. *Introduction to functional programming* (First ed.). Prentice Hall, Englewood Cliffs, New Jersey, USA.

[5] Kathryn Heninger Britton, R. Alan Parker, and David L. Parnas. 1981. A procedure for designing abstract interfaces for device interface modules. In *Proceedings of the 5th international conference on software engineering*, IEEE, San Diego, California, USA, 195–204.

[6] James Church. 2015. *Learning Haskell data analysis.* Packt Publishing Ltd, Birmingham, UK.

[7] H. Conrad Cunningham. 2006. *A programmer's introduction to predicate logic.* University of Mississippi, Department of Computer and Information Science, University, Mississippi, USA. Retrieved from https://john.cs.olemiss.edu/~hcc/csci450/notes/haskell_notes.pdf

[8] H. Conrad Cunningham. 2006. *Notes on program semantics and derivation.* University of Mississippi, Department of Computer and Information Science, University, Mississippi, USA. Retrieved from https://john.cs.olemiss.edu/~hcc/reports/umcis-1994-02.pdf

[9] H. Conrad Cunningham. 2014. *Notes on functional programming with Haskell.* University of Mississippi, Department of Computer and Information Science, University, Mississippi, USA. Retrieved from https://john.cs.olemiss.edu/~hcc/docs/Notes_FP_Haskell/Notes_on_Functional_Programming_with_Haskell.pdf

[10] Roberto Ierusalimschy. 2016. *Programming in Lua* (Fourth ed.). Lua.org, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil.

[11] Samuel N. Kamin. 1990. *Programming languages: An interpreter-based approach.* Addison-Wesley, Boston, Massachusetts, USA.

[12] Donald E. Knuth. 1974. Computer programming as an art. *Communications of the ACM* 17, 12 (1974), 667–673.

[13] Shriram Krishnamurthi. 2007. *Programming languages: Application and interpretation* (First ed.). Brown University, Department of Computer Science, Providence, Rhode Island, USA. Retrieved from http://cs.brown.edu/~sk/Publications/Books/ProgLangs/2007-04-26/

[14] Shriram Krishnamurthi, Benjamin S. Lerner, and Joe Gibbs Politz. 2020. *Programming and programming languages* (Unmaintained final draft ed.). Brown University, Department of Computer Science, Providence, Rhode Island, USA. Retrieved from http://cs.brown.edu/courses/cs173/2012/book/

[15] David L. Parnas. 1972. On the criteria to be used in decomposing systems into modules. *Communications of the ACM* 15, 12 (December 1972), 1053–1058.

[16] Normam Ramsey and Samuel L. Kamin. 2013. *Programming languages: Build, prove, and compare* (Draft ed.). Cambridge University Press, Cambridge, UK.

[17] Michael L. Scott. 2015. *Programming language pragmatics* (Third ed.). Morgan Kaufmann, Waltham, Massachusetts, USA.

[18] Robert W. Sebesta. 1993. *Concepts of programming languages* (Second ed.). Benjamin/Cummings, Boston, Massachusetts, USA.

[19] Peter Sestoft. 2012. *Programming language concepts* (First ed.). Springer, London, UK.

[20] Peter Sestoft. 2017. *Programming language concepts* (Second ed.). Springer, London, UK.

[21] Simon Thompson. 1996. *Haskell: The craft of programming* (First ed.). Addison-Wesley, Boston, Massachusetts, USA.

[22] Simon Thompson. 1999. *Haskell: The craft of programming* (Second ed.). Addison-Wesley, Boston, Massachusetts, USA.

[23] Simon Thompson. 2011. *Haskell: The craft of programming* (Third ed.). Addison-Wesley, Boston, Massachusetts, USA.

[24] E. Peter Wentworth. 1990. *Introduction to functional programming using RUFL*. Rhodes University, Department of Computer Science, Grahamstown, South Africa.