CSci 555 : Functional Programming
Spring Semester 2019
Lecture Notes

H. Conrad Cunningham

Contents

Professor of Computer and Information Science
School of Engineering
University of Mississippi

I maintain these notes as text in the Pandoc's dialect of Markdown using embedded LaTeX markup for the mathematical formulas and then translate the notes to HTML, PDF, and other formats as needed.

Browser Advisory: The HTML version of this document requires use of a browser that supports the display of MathML. A good choice as of August 2020 is a recent version of Firefox from Mozilla.

Lecture Notes

Go To Current Lecture

Class Introduction

1. **(23 Jan)** Discuss class organization and Syllabus: HTML – PDF

Programming Paradigms

2. **(25, 28 Jan)** Survey the *Primary Paradigms* (ELIFP Chapter 2)

    a. Chapter: HTML – PDF

    b. HTML slides: Scala version – Original ELIFP

    c. Scala examples:

    - Imperative example `CountingImp.scala`
    - Imperative example `CountingImp2.scala`
    - Functional example `CountingFun.scala`
    - Procedural example `CountingProc.scala`
    - Modular example `CountingMod.scala`
    - Modular example `CountingMod2.scala`

3. **(25 Jan)** Motivate the study of functional programming

    a. Excerpt from Backus' 1977 Turing Award Address

    b. (for reference) 1977 Turing Award Address: John Backus. Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs, *Communications of the ACM* 21.8 (1978): 613-641.

4. (for reference) Survey the *Object-Based Paradigms* (ELIFP Ch. 3)

    a. Chapter: HTML – PDF

    b. HTML Slides: Original ELIFP

    c. Scala example: Object-Oriented `CountingOO.scala`

    d. Additional notes: Object-Oriented Software Development

    e. Additional slides on Using CRC Cards (Class-Responsibility-Collaboration): HTML slides

- Background: Kent Beck and Ward Cunningham. A Laboratory for Teaching Object-Oriented Thinking, In *Proceedings of the OOPSLA'89 Conference*, ACM, 1989.
- Tutorial: Paul Gestwicki. CRC Card Analysis (YouTube), Ball State University, February 2016.

Basic Scala

5. **(30 Jan, 1, 4, 6 Feb)** Introduce Scala to Java programmers

   a. Note: These Notes were adapted from: Michel Schniz and Philipp Haller, A Scala Tutorial for Java Programmers

   b. *Notes on Scala for Java Programmers*: HTML – PDF

      Diagram for Scala's unified type hierarchy

   c. Scala source code from Notes:

      - `HelloWorld.scala`
      - `FrenchDate.scala` and `MoreDates.scala`
      - `Timer.scala` – `TimerAnonymous.scala`
      - `ComplexNumbers.scala` – `ComplexNumbers2.scala` – `ComplexNumbers3.scala`
      - `ExprCase.scala` – `ExprObj.scala`
      - `OrderedDateTest.scala`

   d. (for reference) Miscellaneous overviews, cheat sheets, and tutorials

      - Tour of Scala
      - Java and Scala Comparison Cheatsheet
      - Concise Java to Scala
      - Learn X in Y Minutes, Where X = Scala

6. **(1 Feb)** Distribute Assignment #1. Due 13 Feb.

Recursive Programming

7. **(6, 8 Feb)** Examine *Recursion Styles, Correctness, and Efficiency: Scala Version*

   a. Notes: HTML – PDF

   b. HTML Slides

   c. Scala source code from Notes: `RecursionStyles.scala`

   d. (for reference) "Tail Call" article on *Wikipedia*

8. **(8 Feb)** Take Quiz #1 over semester content through *Notes on Scala for Java*

9. (for reference) Examine functions adapted from SICP

a. Background reading: Chapter 1 of the classic SICP textbook

   Harold Abelson and Gerald J. Sussman with Julie Sussman. *Structure and Interpretation of Computer Programs*, Second Edition, MIT Press, 1996:

   - book site at MIT Press
   - HTML book
   - SICP ebook site
   - local copy of source code

b. First-order functions in Scala

   - Square root (Newton's Method) version 1 with all public functions: Scala source
   - Square root (Newton's Method) version 2 with nested function definitions: Scala source
   - Factorial — discussed in Recursion Styles notes above: Scala source
   - Fibonacci — discussed in Recursion Styles notes above: Scala source
   - Exponentiation — discussed in Recursion Styles notes above: Scala source
   - Greatest common divisor: Scala source

c. Higher-order functions in Scala

   - Summation (takes function arguments): Scala source

   - Derivative (returns function result): Scala source

Types and Functional Data Structures

10. **(11 Feb)** Survey *Type System Concepts*

    a. Notes: HTML – PDF

    b. HTML slides

11. **(13, 15, 18, 20, 22, 25 Feb, 1 Mar)** Study *Functional Data Structures*

    a. Background reading: "Functional Data Structures," Chapter 3, Paul Chiusano and Runar Bjarnason, *Functional Programming in Scala*, Manning, 2015 (i.e. the Red Book)

    b. Lecture notes on Chapter 3 (primarily on `List` type): HTML – PDF

    c. Scala source: `List2.scala`

    d. Reference: See the discusion of the object-based paradigms above for general information on the object-oriented nature of Scala.

12. **(16 Feb)** Distribute Assignment #2. Originally due 28 February, but extended to 4 March.

13. **(22 Feb)** Take Quiz #2 over *Recursion Styles*, *Type System Concepts*, and Sections 3.1-3.2 of *Functional Data Structures* notes. (Returned on 25 Feb.)

14. **(27 Feb, 1 Mar)** Examine a natural number arithmetic case study

    a. (for reference) Background on Peano arithmetic and design patterns

    - "Peano Axioms" article on Wikipedia
    - "Peano's Axioms" on Wolfram MathWorld
    - Source Making Design Patterns catalog – for information on the Composite, Singleton, and Null Object design patterns used in this case study

    b. Scala versions (2008, 2012, 2019)

    - Functional object-oriented with ordinary classes: Scala source

      This version uses an object-oriented hierarchy of classes/objects organized according to the Composite, Singleton, and Null Object software design patterns. Once created, the Nat objects are immutable. The operations do not modify the state of an object; they create new objects with the modified state. It carries out computations by "passing messages" among the objects, taking advantage of subtype polymorphism (dynamic binding) to associate method calls with the correct implementation. This version also encapsulates the state within the objects in the hierarchy.

    - Functional module with case classes: Scala source

      This version uses a Scala algebraic data type (defined using a sealed trait and case class/objects) with a group of functions defined in the companion object for the trait. It is organized similarly to the List algebraic data type from the Functional Data Structures chapter. This version uses pattern matching to identify the correct operation functionality. By using cases classes/objects, this version exposes the state outside the hierarchy.

    - Functional object-oriented with case classes: Scala source

      This version is in-between the implementations above. It uses the traditional OO structure, but uses case classes/objects instead of ordinary objects. It uses subtype polymorphism for the left argument and uses pattern matching for the right argument to select the correct operation functionality. By using cases classes/objects, this version exposes the state outside the hierarchy.

    c. Elixir version (2015):

- Nat module
- test module
- test script

d. Lua version (2013, 2014): source `nats2.lua`

e. Ruby version (2006): source `Nat.rb`

f. Java version (2004), simpler, no generics:

- abstract base class `Nat`
- subclass `Zero`
- subclass `Succ`
- subclass `Err`
- main program `TestNat`

References: Patterns and Testing

15. (for reference) Survey design patterns

a. Introduction to Patterns: HTML – PDF – HTML slides

b. John Vlissides. Designing with Patterns, In *Pattern Hatching: Design Patterns Applied*, Addison-Wesley, 1998: Powerpoint

c. Patterns notes and slides:

- Pipes and Filters architectural pattern: HTML – PDF – Powerpoint
- Factory Method design pattern (Powerpoint)
- Strategy design pattern (Powerpoint)
- Template Method design pattern (Powerpoint)

d. Additional references:

- Source Making Design Patterns catalog
- Siemens book: Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*, Wiley, 1996.
- "Gang of Four" (GoF) book: Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995. Patterns*, Wiley, 1996.
- Mary Shaw. Some Patterns for Software Architecture, In John M. Vlissides, James O. Coplien, and Norman L. Kerth, editors, *Pattern Languages of Program Design 2*, Addison Wesley, 1996, pages 255-270. – local

16. (for reference) Examine software testing concepts

a. Software Testing Concepts, Chapter 11, *Exploring Language with Interpreters and Functional Programming* (ELIFP): HTML – PDF

b. Testing Haskell Programs, Chapter 12, *Exploring Language with Interpreters and Functional Programming* (ELIFP): HTML – PDF

c. pytest Python testing framework

Functional Error Handling

17. **(4, 6, 8, 18, 20 Mar)** Explore *Handling Errors without Exceptions*

    a. Background reading: "Handling Errors without Exceptions," Chapter 4, Paul Chiusano and Runar Bjarnason, *Functional Programming in Scala*, Manning, 2015 (i.e. the Red Book)

    b. Lecture notes on Chapter 4: HTML – PDF

    c. Scala source code from notes: `Option2.scala` – `Either2.scala`

    d. Scala examples for parameter-passing discussion:

    - Closures: `ClosureExample.scala` – `ClosureExample.out`
    - Thunks: `ThunkExample.scala` – `ThunkExample.out`

    e. Scala example for wrapping exceptions with `Option` and `Either`: `WrapException.scala`

18. **(8, 18 Mar)** Discuss Semester Project

19. **(11-15 Mar) Enjoy Spring Break**

20. **(18, 20 Mar)** Continue discussion of notes on *Handling Errors without Exceptions*

21. **(18, 20, 22 Mar)** Continue discussion of Semester Project; distribute description on 22 March

22. **(25 Mar)** Take *double-point* Quiz #3 over notes on *Functional Data Structures* and *Handling Errors without Exceptions*

23. **(29 Mar)** Return and discuss Quiz #3: `Quiz03Test.scala` – `List3.scala`

Abstract Data Types

24. **(22 Mar)** Distribute description of Assignment #3 (Mealy Machine Simulator); (29 Mar) extend deadline

    See a similar, but imperative and object-oriented, implementation below for the Semantic Model (part e) for the State Machine used for for Fowler's "Secret Panel" Controller External DSLs

25. **(20, 22, 27, 29 Mar; 1, 3, 5 Apr)** Explore *Abstract Data Types in Scala* using the Labeled Digraph ADT case study

    a. Background reading on abstract data types

- "Abstract data type" article on Wikipedia
- Nell Dale and Henry Walker. "Abstract Specification Techniques," Chapter 1, In *Abstract Data Types: Specifications, Implementations, and Applications*, pp. 1-34, D. C. Heath, 1996.
- William R. Cook. Object-Oriented Programming Versus Abstract Data Types, In *Proceedings of the REX Workshop/School on the Foundations of Object-Oriented Languages (FOOL)*, LNCS 489, pp. 151-178, Springer-Verlag, 1990. – local copy
- William R. Cook. On Understanding Data Abstraction Revisited, In *Proceedings of OOPSLA*, October 2009: – local copy

b. Background reading on directed graph ADTs

- Nell Dale and Henry Walker. "Directed Graphs or Digraphs," Chapter 10, In Abstract Data Types: Specifications, Implementations, and Applications, pp. 439-469, D. C. Heath, 1996.
- Conrad Barski. "Building a Text Game Engine,", Chapter 5, In Land of Lisp: Learn to Program in Lisp, One Game at a Time, pp. 69-84, No Starch Press, 2011. The Common Lisp example in this chapter is similar to the classic Adventure game; the underlying data structure is a labeled digraph.

c. Notes on *Abstract Data Types in Scala*: HTML – PDF

d. Scala solutions using method-chaining abstract data type API

- Abstract data type specification and Scala trait (interface):
  ADT interface trait `Digraph`
- List representation for vertices and edges:
  class `DigraphList` – test program `Test_DigraphList.scala`
- Map (HashMap) representation for graph:
  class `DigraphMap` – test program `Test_DigraphMap.scala`

e. (for reference) Haskell solutions

- List representation for vertices and edges:
  module – test module
- Map representation for graph:
  module `DigraphADT_Map` – test module

f. (for reference) Elixir solutions

- Tuple and list representation for graph:
  module – test script

g. (for reference) Notes on Data Abstraction (Parts of these notes have been integrated into the notes on *Abstract Data Types in Scala* above.)

(not updated 2019) Notes on Data Abstraction—Java Supplement (These notes describe nongeneric Java implementations, circa 1997, of the `Stack` and `Day` ADTs specified in the Data Abstraction notes above.)

h. (for reference) Lecture Notes on Modular Design (Parts of these notes have been integrated into the notes on *Abstract Data Types in Scala* above.)

i. (for reference) H. C. Cunningham, Y. Liu, and J. Wang. "Designing a Flexible Framework for a Table Abstraction," In Y. Chan, J. Talburt, and T. Talley, editors, *Data Engineering: Mining, Information, and Intelligence*, pp. 279-314, Springer, 2010.

j. (for reference) Classic papers by David L. Parnas and associates:

   - Abstract interface: Kathryn Heninger Britton, R. Alan Parker, and David L. Parnas. A Procedure for Designing Abstract Interfaces for Device Interface Modules, In *Proceedings of the 5th International Conference on Software Engineering*, pp. 195-204, March 1981.
   - Information hiding: David L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules, *Communications of the ACM*, Vol. 15, No. 12, pp. 1053-1058, 1972.
   - Software families: David L. Parnas. On the Design and Development of Program Families, *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 1, pp. 1-9, March 1976.
   - Extensible system design: David L. Parnas. Designing Software for Ease of Extension and Contraction, *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 1, pp. 128-138, March 1979.
   - Modular specification of large systems: David L. Parnas, P. C. Clements, and D. M. Weiss. The Modular Structure of Complex Systems, *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 3, pp. 259-266, March 1985.

26. **(8 Apr)** Take Quiz #4 over *Abstract Data Types in Scala* above; returned 10 April@. **(10 Apr)** Post Challenge Assignment #4

27. **(10 Apr)** Host guest lecture by J. P. Marum on reactive programming (20 minutes)

28. **(10 Apr)** Distribute Optional Assignment #4

    a. Sandwich DSL Implementation (one opiton on assignment)

       - Scala SandwichDSL Case Study, instructor's lecture notes (with exercises) – Scala code
       - Haskell SandwichDSL Case Study, instructor's lecture notes (with exercises) – Haskell code

Lazy Data Structures

29. **(10 Apr; 1, 3 May)** Explore *Strictness and Laziness*
    a. Background reading: "Strictness and Laziness," Chapter 5, Paul Chiusano and Runar Bjarnason, *Functional Programming in Scala*
    b. Lecture notes on Chapter 5: HTML – PDF
    c. Scala source code from notes: `StreamC.scala`
    d. (for reference) John Hughes, Why Functional Programming Matters, *Computer Journal*, Vol. 32, No. 2, pp. 98-107, 1989.
    e. (for reference) Notes on Functional Program Evaluation Concepts – not updated 2019

Semester Project Presentations

30. **(15 Apr)** Yunshu Wang and Silu Zhang: Spark Fundamentals
31. **(17 Apr)** Nusrat Armin and Carla Rego: PySpark and Jupyter
32. **(22 Apr)** Matthew Toche: Scala Futures and Promises
33. **(24 Apr)** Chris Donelson and Deep Sharma: Squants Package
34. **(26 Apr)** Amulya Arora (and Layton Jones): Software Testing
35. **(29 Apr)** Saman Ray: Akka Actors in Scala
36. **(3 May)** Layton Jones Project Presentation (Software Testing)

End of Semester

37. **(1 May)** Continue exploring *Strictness and Laziness* **above**
38. **(3 May)** Take Quiz #5 over Chapter 5 *Strictness and Laziness*
39. **(6 May: Noon – 3:00 p.m.)** Optional final exam: Contact instructor by Noon on 3 May if you plan to take this exam!
40. **END of Spring 2019 semester:** Enjoy Summer!

More on Abstract Data Types

41. (for reference) Use the Digraph ADT module (to build a game)

    TODO: Create a Scala version of this example.

    a. Wizard's Adventure game, Elixir Version 1

       This game is based on Chapters 5, 6, and 17 of Conrad Barski's *Land of Lisp: Learn to Program in Lisp, One Game at a Time*, No Starch Press, 2011.

Concepts: Importing Elixir modules, use of Elixir features such as atoms, maps, pipes, and complex pattern matching, designing functional programs to handle global state; use of Elixir API modules `Enum`, `Dict`, `List`, `Keyword`, `String`, `IO`, etc.).

b. Wizard's Adventure game, Elixir Version 2

   This version uses a higher-order function to generate game actions and improved handling of the game state.

   Additional concepts: Use of higher-order factory function to generate functions from first- and higher-order arguments; storing functions in a data structure, calling functions stored in a data structure; use of mutator (setter) and accessor (getter) functions for the data structure.

42. (for reference) Examine the CookieJar ADT case study (and other examples)

    a. Cookie Jar ADT Problem Description

    b. Description of Bag Concept (used in specification)

    c. Immutable CookieJar ADT Implementation in Scala (`ICookieJar.scala`) – uses method-chaining functional style with immutable objects

       - ADT specification as Scala trait: Scala source

       - Using Scala List: Scala source

       - Using Scala HashMap: Scala source

       - Using Scala List of Tuples: Scala source

       - Blackbox test script: Scala source

    d. Mutable CookieJar ADT Implementation in Scala (`CookieJar`) – uses object-oriented style with mutable state

       - ADT specification as Scala trait: Scala source
       - Using Scala List: Scala source
       - Using Scala List of tuples: Scala source

       - Using Scala ArrayBuffer: Scala source
       - Using Scala HashMap: Scala source
       - Using Scala array: Scala source
       - Blackbox test script: Scala source

       - Similar specification and Ruby program: text

    e. Carrie's Candy Bowl ADT in Lua – gives the specification of a Candy Bowl ADT (similar to the Cookie Jar above) and two implementation using Lua modules

- Carrie's Candy Bowl ADT Problem Description
- ADT Semantics
- Data representations
- Hashed version `candybowl_hash.lua`
- Unsorted List version `candybowl_list.lua`
- Test driver `test_candybowl.lua`

f. Carrie's Candy Bowl ADT in Scala

- trait `CandyBowl` to define abstract data type interface
- class `CandyBowlList` to define a list-based implementation

g. Older mutable Java abstract data type implementations. These use nongeneric Java implementations done during the 1997-98 timeframe

- Queue ADT – gives a specification for a Queue ADT and implements the ADT directly as two concrete classes
- Ranked Sequence ADT – gives a specification for a Ranked Sequence ADT (similar to `ArrayList` or `Vector`)

NOT FULLY UPDATED

Games

43. (for reference) Examine the Dice of Doom game

a. Background: Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time*, No Starch Press, 2011.

b. Dice of Doom, Elixir Version 1a

This is a basic eagerly evaluated version (similar to that developed on pages 303-325 of *Land of Lisp*). This version supports either two human players or a human player and a simple, minimax search-based "AI" (artificial intelligence) opponent.

c. Dice of Doom, Elixir Version 1b

This is an eager version above with an attempt at memoization of functions `neighbors` and `game_tree` using the Elixir `Agent` modules.

d. Dice of Doom, Elixir Version 2a,

This is a lazy version using Elixir `Stream` data structures (but without the memoization optimizations). This is based on version 1a above plus the discussion from pages 384-389 in chapter 18 of *Land of Lisp*. This version implements a limited depth minimax search, but it does not implement the artificial intelligence heuristics given in the last part of chapter 18.

Domain-Specific Languages

44. (TBD) Discuss domain-specific languages and Sandwich DSL case study

    a. Concepts: domain-specific languages (DSLs); DSLs versus general-purpose programming languages; external versus internal DSLs; shallow versus deep embedding of internal DSLs; use of algebraic data types to implement DSLs

    b. Notes: Domain-Specific Languages

    c. Sandwich DSL Implementation

        - Scala SandwichDSL Case Study, instructor's lecture notes (with exercises) – Scala code
        - Haskell SandwichDSL Case Study, instructor's lecture notes (with exercises) – Haskell code

    d. Sandwich DSL in Lua (similar but not identical to Haskell description

        - Semantic model module `sandwich_model.lua`
        - DSL builder module using function sequence pattern 'sandwich_builder.lua
        - [Test driver] 'test_sandwichDSL.lua](<SandwichDSL/Lua/test_sandwichDSL.lua>

45. (for reference) Discuss State Machine External DSLs based on Martin Fowler's Secret Panel Controller (State Machine) DSLs

    a. Backgrond: Martin Fowler. *Domain Specific Languages*, Addison Wesley, 2011.

        - Fowler introduces the running example, the Secret Panel Controller (State Machine) case study, in DSL Chapter 1)
        - Fowler DSL Chapter 1 "An Introductory Example"
        - Fowler DSL Chapter 2 "Using Domain-Specific Languages"
        - Fowler DSL Chapter 3 "Implementing DSLs"
        - Fowler DSL Chapter 5 "Implementing an External DSL"
        - Fowler DSL Chapter 8 "Code Generation"
        - Fowler DSL Chapter 11 "Semantic Model"
        - Fowler DSL Chapter 17 "Delimiter-Directed Translation"
        - Fowler DSL Chapter 18 "Syntax-Directed Translation"
        - Fowler DSL Chapter 21 "Recursive Descent Parser"
        - Fowler DSL Chapter 21 "Parser Generator"
        - Fowler DSL Chapter 22 "Parser Combinator"
        - Fowler DSL Chapter 51 "State Machine"
        - Fowler DSL Chapter 52 "Transformer Generation"
        - Fowler DSL Chapter 55 "Model Aware Generation"
        - Fowler DSL Chapter 56 "Model Ignorant Generation"
        - Fowler's List of DSL Patterns from DSL book.

    b. Other items from the instructor's notes

- Mealy Machine Description and Exercise gives a formal description of this kind of state machine
- Recursive Descent Parsing gives a set of (Haskell) program templates for recursive descent parsers
- Parsing Combinators describes a preliminary set of (Haskell) parsing combinators for recursive descent parsers
- Wikipedia entry on Parsing Combinator

c. Scala notes

- I developed the Scala versions in 2009 by referring to an in-work version of Fowler's book on his website, so they may not follow his final book precisely. Fowler's code is in Java, Ruby, C#, etc. I updated the Scala programs in 2018 to make sure they would compile and execute under the current Scala release.

d. Graphviz and `dot` language reference and examples

- Note: On my iMacs, I used Homebrew to install the package `graphviz`.
- Graphviz: Graph Visualization Software and the little language/tool `dot`
- `ExprLangModDep.gv` – `script` – `png` output
- `DFA01.gv` – script – `pdf` output
- `NFA01.gv` – script – `svg` output
- `NFA01a.gv` – script – `svg` output
- `Infix1plus1b.gv` – script – `png` output

e. Scala State Machine Semantic Model (Fowler Ch. 1, 3, 11)

- StateMachine.scala
- StateMachineTest.scala
- CommandChannel.scala (mock)
- StateMachineDirect.sh test script

f. Scala XML-based External DSL (Fowler Ch. 1, 3, 5)

- StateMachineXMLTest .scala
- IncrementalStateMachineBuilder.scala
- input file SecretPanel.xml
- StateMachineXML.sh test script

g. Scala Custom External DSL with a Delimiter-Directed parser (Fowler Ch. 1, 3, 5, 17)

- DelimiterDSLTest.scala
- IncrementalStateMachineBuilder.scala
- input file CustomExternalStateMachineDSL.dsl
- DelimiterDSL.sh test script

h. Scala Custom External DSL with hand-coded Ad Hoc Recursive-Descent parser (Fowler Ch. 1, 3, 5, 21)

   - RecursiveDescentTest.scala
   - IncrementalStateMachineBuilder.scala
   - input file CustomExternalStateMachineDSL2.dsl
   - RecursiveDescent.sh test script

i. Scala Custom External DSL (using Scala parser combinator library) with embedded state machine builder (Fowler Ch. 1, 3, 5, 22)

   - CombinatorParserBuilderTest.scala
   - IncrementalStateMachineBuilder.scala
   - input file CustomExternalStateMachineDSL2.dsl
   - CombinatorParserBuilder.sh test script

j. Scala Custom External DSL (using Scala parser combinator library) with full AST construction (Fowler Ch. 1, 3, 5, 22)

   - CombinatorParserASTTest.scala
   - input file CustomExternalStateMachineDSL2.dsl
   - CombinatorParserAST.sh test script

k. Scala Static C Code Generator with Model-Aware target platform library (Fowler Ch. 1, 3, 5, 8, 52, 55)

   - StaticC_GeneratorTest.scala
   - StaticC_Generator.sh test script
   - generated C output file output.c
   - Note: The needed framework code needed to run the generated C program is not yet available in this form. It needs to be reconstructed from Fowler's book.

l. Scala Graphviz Dot Language Code Generator (added 2018) (Fowler Ch. 1, 3, 5, 8, 52)

   - GraphVizCodeGen.scala
   - GraphVizCodeGenTest.scala
   - GraphVizCodeGen.sh test script
   - generated Graphviz Dot output file graph.gv (see test script)
   - Scalable Vector Graphics output from dot graph.svg (see test script)
   - PDF output from dot graph.pdf (see test script)

46. (TBD) Discuss Fowler's Computer Configuration Internal (Ch. 4, 35, 36, 38)

    a. Background: Martin Fowler. *Domain Specific Languages*, (Addison Wesley, 2011).

       - Martin Fowler's List of DSL Patterns from *Domain Specific Languages*, Addison Wesley, 2011

- Fowler DSL Chapter 1 “An Introductory Example”
- Fowler DSL Chapter 2 “Using Domain-Specific Languages”
- Fowler DSL Chapter 3 “Implementing DSLs”
- Fowler DSL Chapter 4 “Implementing an Internal DSL”
- Fowler DSL Chapter 11 “Semantic Model”
- Fowler DSL Chapter 13 “Context Variable”
- Fowler DSL Chapter 32 “Expression Builder”
- Fowler DSL Chapter 33 “Function Sequence”
- Fowler DSL Chapter 34 “Nested Functions”
- Fowler DSL Chapter 35 “Method Chaining”
- Fowler DSL Chapter 36 “Object Scoping”
- Fowler DSL Chapter 37 “Closure”
- Fowler DSL Chapter 38 “Nested Closure”
- Fowler DSL Chapter 39 “Literal List”
- Fowler DSL Chapter 40 “Literal Map”

b. Scala versions (2009)

- Scala semantic model (shared)
- Scala internal DSL using Method Chaining
- Scala internal DSL using Nested Closures and Object Scoping
- CompConfig.sh test script

47. (TBD) Discuss Fowler’s Email Message Building Internal DSL (Ch. 35)

a. Background: Martin Fowler. *Domain Specific Languages*, (Addison Wesley, 2011). See listing of chapters under Computer Configuration internal DSL above.

b. Scala versions (2009)

- Scala internal DSL using Method Chaining and Progressive Interfaces
- test script `EmailProgressive.sh`

48. Explore Fowler’s Lair Configuration DSLs

a. Background on Lair DSL

- Case study: Martin Fowler’s One Lair and Twenty Ruby DSLs, Chapter 3, *The ThoughtWorks Anthology: Essays on Software Technology and Innovation*, The Pragmatic Bookshelf, 2008 – local copy
- Source code: entire Thoughtworks book] [Fowler’s Ruby DSLs
- Patterns (repeated from above): Martin Fowler’s List of DSL Patterns from *Domain Specific Languages*, Addison Wesley, 2011

b. Background on internal DSLs: Martin Fowler. *Domain Specific Languages*, (Addison Wesley, 2011).

See the CSci 658 (Software Language Engineering) course materials for more information about DSLs.

c. Background on Lua

   - See Lua references below
   - Several of the Lair DSL programs must execute with Lua 5.1 because they use a few Lua platform features changed in Lua 5.2 or 5.3. I installed the lua@5.1 package on my iMacs using the Homebrew package manager.
   - At the terminal command line, `lua5.1 testNN.lua` compiles and executes the test driver program for the DSL with files builder program `builderNN.lua` and DSL script `rulesNN.lua`.

d. Shared modules

   Fowler's Ruby source

   - Semantic model (model.rb)
   - Test Driver for semantic model (rules0.rb)

   Lua (2013)

   - Class support module (for implementing classes in Lua)
   - Semantic model
   - Test driver for semantic model (rules00.lua)

   Python 3 (2018)

   - Semantic model
   - Test driver for semantic model (rules00.py)

e. Internal DSL using Global Function Sequence pattern

   Fowler's Ruby

   - builder module (builder8.rb)
   - dsl script (rules8.rb)

   Lua (2013)

   - builder module (builder08.lua)
   - dsl script (rules08.lua)
   - test driver (test08.lua)

   Python 3 (2018)

   - builder module (builder08.py)
   - direct execution test of DSL script (rules08x.py)
   - dynamically loaded dsl script (rules08.py)
   - test driver for dynamically loaded dsl script (test08.py)

f. Internal DSL using Class Method Function Sequence and Method Chaining patterns

   Fowler's Ruby

   - builder module (builder11.rb)
   - dsl script (rules11.rb)

   Lua (2013)

   - builder module (builder11.lua)
   - dsl script (rules11.lua)
   - test driver (test11.lua)

   Python 3 (2018)

   - builder module (builder11.py)
   - dynamically loaded dsl script (rules11.py)
   - test driver for dynamically loaded dsl script (test11.py)

g. Internal DSL using Expression Builder and Method Chaining patterns

   Fowler's Ruby

   - builder module (builder14.rb)
   - dsl script (rules14.rb)

   Lua (2013)

   - builder module (builder14.lua)
   - dsl script (rules14.lua)
   - test driver (test14.lua)

   Python 3 (2018)

   - builder module (builder14.py)
   - dynamically loaded dsl script (rules14.py)
   - test driver for dynamically loaded dsl script (test14.py)

h. Internal DSL using Nested Closures pattern

   Fowler's Ruby

   - builder module (builder3.rb)
   - dsl script (rules3.rb)

   Lua (2013)

   - builder module (builder03.lua)
   - dsl script (rules03.lua)
   - test driver (test03.lua)

   Python 3 – none yet. (Python's weak syntactic support for lambdas does not allow the relatively direct approach usable in Ruby, Scala, and Lua.)

i. Internal DSL using Expression Builder, Object Scoping, and Method Chaining patterns

   Fowler's Ruby

   - builder module (builder17.4b)
   - dsl script (rules17.rb)

   Lua (2013)

   - builder module (builder17.lua)
   - dsl script (rules17.lua)
   - test driver (test17.lua)

   Python 3 – none yet

j. Internal DSL using Literal Collection pattern

   Fowler's Ruby

   - builder module (builder22.rb)
   - dsl script (rules22.rb)

   Lua (2013)

   - builder module (builder22.lua)
   - dsl script (rules22.lua)
   - test driver (test22.lua)

   Python 3 – none yet

k. External DSL using Parser/Builder (no corresponding example in Fowler book chapter)

   Lua (2013)

   - Note: This program requires installation of a Lua LPEG library via luarocks. It must be compatible with whatever version of Lua is being used.
   - builder module (builderLPEG1.lua)
   - dsl script (rulesLPEG1.dsl)
   - test driver (testLPEG1.lua)

   Python 3 (2018)

   - Note: This program requires installation of the Python 3 package Parsita, a parser combinator library similar to Scala's
   - builder module (builderParsita1.py)
   - dsl script (rulesParsita1.dsl)
   - test driver (testParsita1.py)

49. (TBD) Discuss Fowler's DSL Reader framework (2006)

    a. Background:

- Martin Fowler. Language Workbenches: The Killer-App for Domain Specific Languages? June 2005.
- Martin Fowler. Generating Code for DSLs, June 2005.

b. Ruby shared modules (2006)

- DSL Reader Framework module (ReaderFramework.rb)
- DSL Reader Utilities mix-in module (ReaderUtilities.rb)
- Data input file (fowlerdata.txt)
- Text DSL description (dslinput.txt)
- XML DSL description (dslinput.xml)

c. Ruby direct configuration and testing of Reader (2006)

- BuilderDirect.rb

d. Ruby single-pass external text DSL (2006)

- TextSinglePass.rb

e. Ruby two-pass external XML DSL (2006)

- TwoPass.rb
- class BuilderExternal source code generated by TwoPass.rb

f. Ruby internal DSL

- RubyDSL.rb

50. H.C. Cunningham. "A Little Language for Surveys: Constructing an Internal DSL in Ruby," In *Proceedings of the ACM SouthEast Conference*, 6 pages, March 2008.

    a. manuscript

    b. presentation

    c. Ruby source

    d. test DSL input file

    e. test DSL input file with errors

Object-Oriented Programming and Frameworks

51. (for reference) Simple, silly Employee hierarchy example

    a. Scala (2008, 2010)

    b. Ruby (2006)

52. (TBD) Overview object-oriented programming in Scala

    a. Instructor's notes on Object-Oriented Software Development

b. Scala translation of Frog dynamic composition example: Scala source – output

c. Modified Philosophical Frog example from Odersky et al: Scala source – output

d. Modified Stackable traits example (IntQueue) from Odersky et al: Scala source – output

53. (TBD) Frameworks, based on Timothy Budd's An Introduction to Object-Oriented Programming, Third Edition, Chapter 21

    a. Simple Sorting Framework examples.

    - Low-level concrete Employee sorting: Scala source
    - Insertion Sorting framework: Scala source
    - Employee Sorting application of framework: Scala source
    - Test code for Employee Sorting application of framework: Scala source

    b. Ice Cream Store discrete event simulation.

    - Simulation framework: Scala source
    - Ice Cream Store application: Scala source typical output

54. (TBD) Cunningham group "Using Classic Problems ... " paper.

    a. H. C. Cunningham, Y. Liu, and C. Zhang. "Using Classic Problems to Teach Java Framework Design," *Science of Computer Programming*, Special Issue on Principles and Practice of Programming in Java (PPPJ 2004), Vol. 59, No. 1-2, pp. 147-169, January 2006. doi: 10.10.16/j.scico.2005.07.009. preprint PDF

    b. Related tutorial: H. C. Cunningham, Y. Liu, and C. Zhang. "Teaching Framework Design Using Classic Problems," *Journal of Computing Sciences in Colleges*, Vol. 21, No. 5, pp. 10-12, CCSC, May 2006: abstract presentation

55. (TBD) Scala Divide-and-Conquer Framework, similar to the Java framework in the "Using Classic Problems ... " paper

    a. Template-based Divide-and-Conquer Framework (DivConqTemplate): Scala source

    b. Strategy-based Divide-and-Conquer Framework (DivConqStrategy): Scala source

    c. Traits for Problem and Solution descriptions for both frameworks (DivConqProblemSolution):
    Scala source

    d. Application of Template-based framework to QuickSort (QuickSort-TemplateApp):

Scala source – program output

e. Application of Strategy-based framework to QuickSort (QuickSort-StrategyApp):
Scala source – program output

f. Descriptor for QuickSort state for both QuickSort applications (QuickSortDesc):
Scala source

56. (TBD) Scala Binary Tree Framework, similar to the Java framework in the "Using Classic Problems . . . " paper

    a. Straightforward translation of the Java program to Scala:

        - Top-level framework (BinTreeFramework): Scala source
        - Second-level Euler tour framework (EulerTourVisitor): Scala source
        - Second-level mapping framework (MappingVisitor): Scala source
        - Second-level breadth-first framework (BreadthFirstVisitor): Scala source
        - Application of BinTree frameworks (BinTreeTest): Scala source

    b. Generic implementation of BinTree framework in Scala:

        - Top-level framework (BinTreeFramework): Scala source
        - Second-level Euler tour framework (EulerTourVisitor) Scala source
        - Second-level mapping framework (MappingVisitor): Scala source
        - Second-level breadth-first framework (BreadthFirstVisitor): Scala source
        - Application of BinTree frameworks (BinTreeTest): Scala source

Programming Language Reference Materials

57. Free online programming language textbooks and tutorials

Haskell

58. Learn X in Y Minutes, Where X = Haskell

59. H. Conrad Cunningham. Introduction to Functional Programming Using Haskell, Computer and Information Science, University of Mississippi, 2016-2017. (I am writing this "textbook" primarily for CSci 450 and likely will use the title *Exploring Languages with Interpreters and Functional Programming* for the Fall 2018 draft.)

60. H. Conrad Cunningham. *Notes on Functional Programming with Haskell*, Computer and Information Science, University of Mississippi, 1994-2014. (I am revising this document and integrating it with other materials to

produce the new "textbook" above *Exploring Languages with Interpreters and Functional Programming*. The chapters at the chapters at the end have not yet been integrated.)

61. Haskell language website

    a. *Haskell 2010 Language Report*
    b. *GHC User's Guide*

62. *Haskell Cheat Sheet*

63. School of Haskell online tutorials

64. Miron Lipovaca. *Learn You a Haskell for Great Good: A Beginner's Guide*, a free online tutorial at http://learnyouahaskell.com/. (It is also in print from No Starch Press, 2011.)

65. Kees Doets and Jan van Eijck. *The Haskell Road to Logic, Math and Programming*, March 2004. This book is also available in print, published by College Publications, 2004.

66. Paul Hudak. *The Haskell School of Music: From Signals to Symphonies*, Version 2.6, January 2014. This book is a recent rewrite of Hudak's *The Haskell School of Expression: Learning Functional Programming through Multimedia*, Cambridge University Press, 2000.

67. Simon Marlowe. *Parallel and Concurrent Programming in Haskell: Techniques for Multicore and Multithreaded Programming*,

    2013. This book is also available in print, published by O'Reilly Media, 2013.

68. Bryan O'Sullivan, Don Stewart, and John Goerzen. *Real World Haskell*,

    2008. This book is also available in print, published by O'Reilly Media in November 2008.

Lua

69. Learn X in Y Minutes, Where X = Lua

70. Definitive reference: Roberto Ierusalimshcy. *Programming in Lua*, Fourth Edition, Lua.org, Rio de Janiero, Brazil, 2016. (The First Edition of this book, covering Lua 5.0, is available online at https://www.lua.org/pil/co ntents.html.)

71. Instructor's Lua slides (from CSci 450, Fall 2016)

    a. Background: The slides below are adapted, in part, from *Programming in Lua, Slides for Course* by Fabio Mascarenhas from the Federal University of Rio de Janeiro, Brazil. He taught a course, which used Lua 5.2, at Nankai University, P. R. China, in July 2013.

b. Introduction to Lua HTML slides based, in part, on Mascarenhas slide sets 0-5
c. Advanced Lua Functions HTML slides based, in part, on Mascarenhas slide set 6
d. Modules in Lua HTML slides based, in part, on Mascarenhas slide set 9
e. Lua Metatables HTML slides based, in part, on Mascarenhas slide set 10
f. Lua Objects HTML slides based, in part, on Mascarenhas slide set 11

Python 3

72. Learn X in Y Minutes, Where X = Python3
73. Bernd Klein. Python Course

Ruby

74. Learn X in Y Minutes, Where X = Ruby
75. Chris Pine. Learn to Program
76. why the lucky stiff (Jonathan Gillette). why's (poignant) guide to Ruby

Scala

77. Learn X in Y Minutes, Where X = Scala
78. Definitive reference: Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala*, Third Edition, Artima, 2016. The first edition (2008) is available online at http://www.artima.com/pins1ed/.
79. Martin Odersky. *Scala by Example*, EPFL, 2014. [local]
80. Instructor's Notes on Scala for Java Programmers
81. Paul Chiusano and Runar Bjarnason. *Functional Programming in Scala*, Manning, 2015.
    a. Instructor's Notes on Functional Data Structures (Chapter 3)
    b. Instructor's Notes on Error Handling without Exceptions (Chapter 4)
    c. Instructor's Notes on Strictness and Laziness (Chapter 5)
