Expression Tree Calculator

H. Conrad Cunningham

27 April 2022

Contents

Expression Tree Calculator **1**
Scala Versions (2008-19) . . . 1
Python versions (2018) . . . 1
Object-oriented versions . . . 1
Functional module versions . . . 2
Expression tree parser . . . 2
Lua Versions (2013-16) . . . 2
Recursive function versions (2013, 2014) . . . 2
Object-oriented versions (2013, 2016) . . . 2
LPEG parsers (2013) . . . 2
Haskell Version (2017) . . . 3
Acknowledgements . . . 3
References . . . 3

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.

Expression Tree Calculator

Scala Versions (2008-19)

These Scala programs are from the chapter *Notes on Scala for Java Programmers* [1] (as HTML) (as PDF).

- Recursive function version using case classes `ExprCase.scala`
- Traditional object-oriented version `ExprObj.scala`

Python versions (2018)

Object-oriented versions

- Basic OO version `ExprOO2.py`
- OO versions using abstract base classes

  TODO: In the following, I seem to have been exploring various OO approaches. These need to be reexamined to determine which should be included here. In some of these, I was also exploring dataclasses and type annotations.

  - Using inheritance `ExprABC2.py`
  - Using registration and class methods `ExprABC_RegClassmeth2.py`
  - Using registration and mixin `ExprABC_RegMixin2.py`
  - Using registration and delegation `ExprABC_RegDelegate2.py`
  - Using dataclass decorators and type annotations `ExprDC.py`

Functional module versions

- Basic module of functions `ExprFuncMod2.py`
- Using dataclass decorators and type annotations `ExprFuncMod2.py`
- Using table (dictionary) of functions `ExprEvalTab.py`

Expression tree parser

These programs require the Python package Parsita, a parser combinator library similar to Scala's.

TODO: These are not complete. At least a test driver is needed.

- Abstract syntax tree `calc_ast2.py` adapted and extended to work with Calculator expression parser – uses frozen data classes
- Parser `calc_parser.py` – uses Parsita combinator-based parser for simple Calculator-like language including function calls

Lua Versions (2013-16)

Recursive function versions (2013, 2014)

- Lua Recursive Functions with Record Representation `exprRecFuncRecord.lua`

- Lua Recursive Functions with List Representation `exprRecFuncList2.lua`
- Lua Evaluation Function Table with List Representation `exprEvalTable2.lua`

Object-oriented versions (2013, 2016)

- Lua Prototype Object-Based `exprObjBased.lua`
- Lua Object-Oriented with Inheritance

LPEG parsers (2013)

These programs require installation of a compatible LPEG library.

- Parser with captures `exprParser.lua`
- Parser with semantic actions `exprParserSemantic.lua`

Haskell Version (2017)

- Expression Tree Calculator case study
  - as HTML
  - as PDF
  - Haskell source

Acknowledgements

What I call the "recursive function version using case classes" is more or less the first version of these examples. It is closely based on the example in an early version of the Schinz and Haller tutorial [3]. I modified that version and also created the "traditional object-oriented version" for an assignment in the prototype offering of the Multiparadigm Programming course. I subsequently developed the Lua versions in 2013, the Haskell version in 2017, and the Python versions in 2018 for that or other courses. The Expression Tree is one of the examples or assignments I redesign and implement when I am learning and teaching a new language or want an assignment of that nature.

This example has been more or less expanded into the ELI Calculator Language introduced in several chapters of *Exploring Languages with Interpreters and Functional Programming* (ELIFP) [2].

I retired from the full-time faculty in May 2019. As one of my post-retirement projects, I am continuing work on possible textbooks based on the course materials I had developed during my three decades as a faculty member. In January 2022, I began refining the existing content, integrating separately developed materials together, reformatting the documents, constructing a unified bibliography (e.g., using citeproc), and improving my build workflow and use of Pandoc.

I maintain this chapter 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.

References

[1] H. Conrad Cunningham. 2019. *Notes on Scala for Java programmers*. University of Mississippi, Department of Computer and Information Science, University, Mississippi, USA. Retrieved from https://john.cs.olemiss.edu/~hcc/docs/ScaldFP/ScalaForJava/ScalaForJava.html

[2] H. Conrad Cunningham. 2022. *Exploring programming languages with interpreters and functional programming (ELIFP)*. University of Mississippi, Department of Computer and Information Science, University, Mississippi, USA. Retrieved from https://john.cs.olemiss.edu/~hcc/docs/ELIFP/ELIFP.pdf

[3] Michel Schinz and Phillipp Haller. 2016. A Scala tutorial for Java. Retrieved from https://docs.scala-lang.org/tutorials/scala-for-java-programmers.html