CSI 3350: Programming Languages

Credits Hours: 4 credits, 3.57 contact hours/week.

Instructor: Hua Ming, Ph.D.

Text book:

1. ESSENTIALS OF PROGRAMMING LANGUAGES – 3rd Edition Publisher: The MIT Press; (April 18, 2008) ISBN: 978-0262062794
2. THE LITTLE SCHEMER – 4th Edition Publisher: The MIT Press; (December 21, 1995) ISBN: 978-0262560993
3. STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS – 2nd Edition (URL: http://deptinfo.unice.fr/~roy/sicp.pdf)

Specific course information

CSI 3350 is about the principles, design and implementation of modern programming languages. Generally speaking, syntax, semantics and their intimate interplay in modern programming languages are the major aspects of study in CSI 3350. Offered fall.
The major approach we use to study programming languages in CSI 3350 is experiment-based. In other words, you will get your hands wet and build an interpreter for a non-trivial programming language. To achieve it within one short semester, we will approach it by studying functional programming paradigm, which is a hot area currently in the industry. Python and Scala, for example, embody this trend.

Prerequisites: CSI 2310 and MTH 2775 and major standing

Required course for CS major

Course Objectives: Upon successful completion of this course, students should be able to

- Describe main quality criteria for high level programming languages such as readability, writability, reliability and cost [ABET CS: (b)]
- Describe syntax of fundamental program components [ABET CS: (i)]
- Discuss fundamental concepts of operational semantics [ABET CS: (a, i)]
- Describe activation records, parameter passing and access to nonlocals [ABET CS: (i)]
- Describe data types and type systems [ABET CS: (i)]
- Explain and apply major features of functional and logic programming languages [ABET CS: (a, i)]

List of Topics:

- Introduction, patterns of programming and software setup
- Functional programming paradigm and Scheme language
- Flat recursion, higher order procedures, binding and environment

- Syntax and Data Abstraction
- Data Type definition and representation
- Using define-datatype: a tool to avoid grunt work {Exam One}
- Context Free Grammar based parser generator
- Introducing SLLGEN: a LL(1) parser generator
- Building a parser for a simple programming language with a given grammar
- Expand the language to embrace more language features
- Further expand the language to include procedure feature {Exam Two}
- Keep expanding the language to include recursive procedures
- Type-checking and type inference