**Basic Course Information:** CS 5400

Course Title: Topics in Compiler Design

Units: 3 units

CS number: C-2

Component: Lecture

Instructional Mode: Face-to-Face and web-assisted

Grading Basis: Graded only

Repeated Basis: May be taken only once

Cross listed Course: N/A

Dual-listed Course: N/A

Major course/Service course/GE course: Major course

Date Prepared: March 31, 2015

Prepared by: Yu Sun

# I. Catalog Description

Code generation and optimization. Data flow analysis. Syntax-directed translation. Parallelizing compiler. Automatic compiler generation.

**II. Required Coursework and Background**

Pre-requisite(s): CS 4110 or consent of instructor.

# III. Expected Outcomes

On successful completion of this course, students will be able to:

1. Gain fluency in describing the theory and practice of compilation, in particular, the lexical analysis, syntax, and semantic analysis, code generation and optimization phases of compilation.
2. Create lexical rules and grammars for a programming language.
3. Use Flex or similar tools to create a lexical analyzer and Yacc/Bison tools to create a parser.
4. Implement a lexer without using Flex or any other lexer generation tools.
5. Implement a parser such as a bottom-up SLR parser without using Yacc/Bison or any other compiler-generation tools.
6. Implement semantic rules into a parser that performs attribution while parsing.
7. Design a compiler for a concise programming language.

Outcomes of this course will build student capacity in each of the following areas as defined by programmatic objectives for the computer science major.

P-SLO 3. An ability to build applications, either individually or in a team, that are robust, reliable, and maintainable.

P-SLO 4. A breadth of advanced knowledge and skills in applied areas of computer science.

# IV. Instructional Materials

Text:

1. Aho, Lam, Sethi, Ullman. Compiler Design: Principles, Techniques, and Tools (Addison Wesley, 2nd Edition). ISBN: 978-0321486813.

# V. Minimum Student Material

Textbook and class handouts

# VI. Minimum College Facilities

A classroom with a projection system, a computer laboratory

# VII. Course Outline

1. Introduction to Compiler Design
2. Lexical Analysis, such as NFA/DFA conversion, Flex, etc.
3. Syntax Analysis, such as recursive-descent parsing, LL (1) grammar and predictive parsing, LR (0) parsing and SLR (1) parsing, LR (1) parsing and LALR (1) parsing, Yacc, etc.
4. Syntax-Directed Translation (SDT), such as SDT with LL-parser and DAG
5. Type Checking
6. Run-Time Environments
7. Intermediate Code Generation
8. Compiler Optimization

# VIII. Instructional Methods

Lecture

Problem-solving/Discussion

In-class exercises

Project-based learning

# IX. Evaluation of Outcomes

A. Student Assessment

1. Homework assignments

2. Projects

3. Midterm exam

4. Final exam

B. Meaningful Writing Assignment

In courses that largely involve software design and implementation, students are expected to explain and document their designs. Good documentation is essential to software longevity, and good written communication (both in and beyond programming code) is necessary for good documentation. The project provides a meaningful and significant writing component, and supports practically all of the course learning outcomes, as shown in the following matrix.

C. A Matrix of Course Student Learning Outcomes vs Methods of Assessment

If the course is being evaluated for accreditation purposes, approved department accreditation assessment tools will additionally be utilized.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Course Learning Outcomes | Methods of Assessment | | | |
| Homework Assignments | Projects | Midterm Exam | Final Exam |
| Gain fluency in describing the theory and practice of compilation, in particular, the lexical analysis, syntax, and semantic analysis, code generation and optimization phases of compilation. | x | x | x | x |
| Create lexical rules and grammars for a programming language. | x | x | x |  |
| Use Flex or similar tools to create a lexical analyzer and Yacc/Bison tools to create a parser. |  | x | x |  |
| Implement a lexer without using Flex or any other lexer generation tools. |  | x | x |  |
| Implement a parser such as a bottom-up SLR parser without using Yacc/Bison or any other compiler-generation tools. | x | x |  | x |
| Implement semantic rules into a parser that performs attribution while parsing. | x | x |  | x |
| Design a compiler for a concise programming language. | x | x | x | x |