**CS440 - Programming Languages and Translators (3 credit hours)**

**Course Manager** – Dr. Xiang-Yang Li, Assistant Professor

3 credit hours; required for CS, elective for CPE; 150 min. lecture each week

**Current Catalog Description** - Study of commonly used computer programming languages with an emphasis on precision of definition and facility in use. Scanning, parsing, and introduction to compiler design. Use of compiler generating tools. Prerequisite: CS 330, CS 351. (3-0-3)

**Textbook**

**References** - other textbooks or materials
- Ravi Sethi, *Programming Languages*, 2nd edition, Addison-Wesley

**Course Goals** - Students should be able to:
- Summarize the evolutionary history of programming languages and illustrate how this history has led to the paradigms available today. Students are exposed to various programming languages.
- Evaluate the tradeoffs between the different paradigms, considering such issues as space efficiency, time efficiency (of both the computer and the programmer), safety, and power of expression.
- Describe the differences between compiled and interpreted execution models, outlining the relative merits of each.
- Outline the phases involved in language translation.
- Explain how abstraction mechanisms support expressiveness of programming languages.
- Describe the difference between call-by-value and call-by-reference parameter passing.
- Demonstrate different forms of binding, visibility, scoping, and lifetime management.
- Defend the importance of types and type checking in providing abstraction and safety.
- Defend the importance of abstractions, especially with respect to programming-in-the-large.
- Describe the steps and algorithms used by language translators.
- Recognize the underlying formal models such as finite state automata, pushdown automata, and their connection to language definition through regular expressions and grammars.
- Discuss the effectiveness of optimization.
- Explain the impact of a separate compilation facility and the existence of program libraries on the compilation process.
- In addition, as elective, the students may also need to know
  - Outline the strengths and weaknesses of the functional programming paradigm.
  - Design, code, test, and debug programs using the functional paradigm.
  - Explain the use of functions as data, including the concept of closures.
  - Outline the philosophy of object-oriented design and the concepts of encapsulation, subclassing, inheritance, and polymorphism.
  - Design, code, test, and debug simple programs in an object-oriented programming language.
  - Describe runtime implementation techniques for objects and classes.

**Prerequisites by Topic**
- Experience writing basic programs in any computer language or a strong mathematics background.

**Major Topics Covered in Course**
1. Language Design. 3
2. Compilation and Interpretation. 3
3. Programming Language Syntax and Prolog 3
4. Prolog and Automata Theory 2
5. Push-Down Automata 3
6. Names, Scopes and Bindings 3
7. Semantic Analysis 3
8. Control Flow 2
9. Sequencing, Selection, Iteration and Recursion 3
10. Data Types 3
11. Subroutines 3
12. Exception handling and coroutines 2
13. Data Abstraction 3
14. Concurrency 6

Course Admin, Quiz #1, #2, Midterm Exam #1 3 hours
Final Exam - 45 hours

Laboratory projects (specify number of weeks on each)

- The course had 4 programming assignments. (2 weeks for each programming assignment)
  - The first two assignments include writing simple programs in Icon programming language in order to make them have a sense of the programming language.
  - The third assignment involves using a TCLL1 parser, which is written in Icon. Given a badly constructed grammar, they have to apply some rules to make the grammar a good one and pass them through the TCLL1 parser to test the results.
  - The fourth assignment is that they have to modify the Euler Compiler, written in Euler language, to establish some required goals.
  - These assignments give them good insights and basic foundations for constructing grammars from the scratch. The students learn practical usage of these in addition to the theoretical study of topics about grammar rules and state machines in the classroom.

Estimate CSAB Category Content in Credit Hours

<table>
<thead>
<tr>
<th>CORE</th>
<th>ADVANCED</th>
<th>CORE</th>
<th>ADVANCED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structures</td>
<td>0</td>
<td>0</td>
<td>Computer Organization and Architecture</td>
</tr>
<tr>
<td>Algorithms</td>
<td>0</td>
<td>0</td>
<td>Concepts of Programming Languages</td>
</tr>
<tr>
<td>Software Design</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Oral and Written Communications - Every student is required to submit at least __0___ written reports (not including exams, tests, quizzes, or commented programs) of typically __0___ pages and to make __0___ oral presentations of typically __0___ minutes duration. Include only material that is graded for grammar, spelling, style, and so forth, as well as for technical content, completeness, and accuracy.

- none

Social and Ethical Issues - Please list the topics that address the social and ethical implications of computing covered in all course sections. Estimate the class time spent on each topic. In what ways are the students in this course graded on their understanding of these topics (e.g., test questions, essays, oral presentations, and so forth)?

- none

Theoretical Foundations - Please list the types of theoretical material covered, and estimate the time devoted to such coverage in contact (lecture and lab) hours.

- Grammar construction (LL1), 3 hours
- Cleaning up bad grammars, 2 hours
- Automata theory, 6 hours
- Recursion, 3 hours
- Backtracking, 3 hours

Problem Analysis - Please describe the problem analysis experiences common to all course sections.
- Most programming assignments include solving problem starting with grammar construction, usage of parsing tools, and modification of compiler code to determine the program output.
- Cleaning up some bad grammars and parsing them are two of the main emphasis.

Solution Design - Please describe the design experiences common to all course sections.
- There are 2 large-scale assignments (2 weeks each). Major object is to integrate various concepts learned in the class. The objective of the first project is to write a prolog program for taking derivatives. The students have to write some differentiation rules in such a way that the program can deal with, sine function, cosine function, logarithm function (ln(x)), the inverse (under composition) of the previous functions, polynomials, and the operations (sum, subtraction, division, multiplication and exponentiation) among them.

Other Course Information
- Additional Suggested Course Assignments
  - 2 quiz exams (100 minutes, around 70% programming)
  - 1 midterm exam (100 minutes, around 70% programming)
  - 1 final exam (120 minutes, around 70% programming)
- Planned Course Enhancements
  - New course goal: This course teaches advanced principles of computer programming language design. It includes major programming language paradigms in a unified framework, providing a sound practical and theoretical basis for programming language design. It emphasizes the view that programs and languages are mathematical structures amenable to rigorous analysis. (Fall 2002)
  - New course catalog description: Introduction to a myriad of programming languages, their structure, and their implementation. Basic language design principles; abstract data types (lists, arrays, user-defined types); functional languages; type systems; object-oriented languages. Using imperative and functional programming as unifying themes, major language design paradigms will be explored. Prerequisite: CS 330, CS 351. (3-0-3) (Fall 2002)
  - Introduce more programming languages to the class to make students be exposure to a myriad of programming languages. (Fall 2002)
  - Although the topics on grammars are present in the current textbook for cs440, it’s not given enough relevance in the course syllabus. Practical approach to these topics and making students use interesting parsing tools and compilers would help them gain more insights into the topics such as grammars. (Spring 2003)
  - Introduce a new 400 level compiler course that introduces the principles, techniques, and tools for the design and construction of compilers for modern programming languages. It will cover topics such as grammar construction, parsing, compiler structure, syntax analysis, syntax-directed translation, compiler construction tools, semantic analysis, intermediate representation, code generation, optimization techniques. (Fall 2003)