CSE 4713
Programming Languages

REQUIRED/ELECTIVE:
Computer Science: Required
Software Engineering: Elective
Computer Engineering: Elective

CATALOG DATA:
CSE 4713/6713. Programming Languages. (3) (Prerequisites: CSE 3124 or ECE 3724 and CSE 3813 both with a grade of C or better). Three hours lecture. An introduction to programming language specification and analysis. Additional topics include control structures, data types and structures, runtime environments, binding strategies, compilers, and interpreters.

PREREQUISITE BY TOPIC:
1. Program development using one or more high-level programming languages.
2. Object-oriented problem analysis and program development using a language with object-oriented support.
4. Formal languages and automata theory, with emphasis on grammars and automata for context-free languages.

TEXTBOOKS AND OTHER REQUIRED MATERIAL:

COORDINATOR:
Julian E. Boggess, Ph.D.

COURSE OBJECTIVES:
The major goal of this course is to develop a view of programming languages not only as means to solve problems but as subjects of analysis and design. To achieve that goal, we will take a broad look at a variety of programming language implementation issues, methods, and implications. Next, we will focus on some specific programming language design features and issues, and examine their implications. Finally, we will investigate the functional, logic, and major imperative paradigms and languages as alternatives to the object-oriented ones.

It is expected that, by the end of the course, the student will be able to:
1. effectively apply concepts from prerequisite courses, especially formal languages and architecture courses, in the context of evaluating the features of programming languages.
2. explain and evaluate design and implementation features of programming languages.
3. apply conceptual knowledge of the syntax of languages, as well as the design of language data structures and control statements, to the efficient implementation of a working language.
4. test his or her implementation solution sufficiently well to have a reasonable level of confidence that the solution will survive unknown tests which will be applied to it
5. demonstrate comprehension of short programs written in functional and logic paradigms

TOPICS COVERED:
1. Introduction; history of programming language development; review of formal languages and automata, representing language syntax and semantics, application to design and implementation of modern languages. 13
2. Binding, type checking, scope, lifetime, referencing environments; design and implementation of primitive data types, structured data types, dynamically allocated space 7
3. Expressions, assignment statements, and flow-of-control statements, contrasted and compared across languages. 4
4. Issues related to procedures and functions: Implementation of dynamic and static scope, semantics of subprogram call/return paradigms, design and implementation of subprogram call/return paradigms. Runtime stack, activation records.
5. Functional and logic programming.
6. Other issues: I/O, error handling, concurrent programming, etc.

CONTRIBUTION TO PROFESSIONAL COMPONENT:
Engineering Topics

ASSESSMENTS:
1. 3 in-class tests. Tests include discussion questions; students are expected to synthesize their previous experience from the CS curriculum in their written discussions.
2. 3 quizzes.
3. 5 programming projects, each with a laboratory report:
   a. Write a lexical analyzer to recognize tokens in a small working language. (2 weeks)
   b. Write an interpreter to recognize declarations, build an appropriate symbol table, and execute assignment statements for int and float variable types, with mixed mode and cascaded assignments. Simulate memory separately from the symbol table. (3 weeks)
   c. Draw the syntax graphs and write an LL(1) grammar for the language, including arithmetic, I/O, and control statement syntax. The grammar must reflect appropriate precedence and associativity (1 week)
   d. Expand the interpreter to include arithmetic with standard precedence and associativity, unary operators such as --, ++ and --, I/O, branching and looping, with appropriate manipulation of simulated memory. (4 weeks)
   e. Expand the interpreter to include functions or multi-dimensional arrays. (1-2 weeks)

RELATIONSHIP TO PROGRAM OUTCOMES:
Note: Parenthesized list indicates the ABET and Software Engineering outcomes addressed by each criteria.
1. The student will be able to apply concepts from prerequisite courses, especially formal languages and architecture courses, effectively in the context of evaluating the features of programming languages (se7, abet a)
2. The student will be able to explain and evaluate design and implementation features of programming languages. (se7, abet g)
3. The student will be able to apply conceptual knowledge of the syntax of languages, as well as the design of language data structures and control statements, to the efficient implementation of a working language. (se3, abet a, b, and c)
4. The student will be able to test his or her implementation solution sufficiently well to have a reasonable level of confidence that the solution will survive unknown tests which will be applied to it (abet b and c)
5. The student will demonstrate comprehension of short programs written in functional and logic paradigms (abet k)

PREPARED BY:
Julian Eugene Boggess, III; Associate Professor
Department of Computer Science and Engineering
18 April 2005
ESTIMATE CSAB CATEGORY CONTENT:

<table>
<thead>
<tr>
<th>CATEGORY</th>
<th>CORE</th>
<th>ADVANCED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computer Organization and Architecture</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Concepts of Programming Languages</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

ORAL AND WRITTEN COMMUNICATIONS:

No material in this course is graded for grammar, spelling, or style. No oral reports are presented.

SOCIAL AND ETHICAL ISSUES:

Social and ethical issues may be discussed in any given semester in classroom discussion. However, they are not a formal part of the syllabus, nor are they a guaranteed component of any particular semester's discussion.

THEORETICAL CONTENT:

Formal methods for specifying syntax and semantics; binding and scoping strategies; data and control abstractions (33%) 

PROBLEM ANALYSIS:

The five-part semester's project is focused on the analysis of the formal and practical requirements to support a small working language, from the design of the grammar through the manipulation of a symbol table and memory to implement arrays. Students must do a substantial amount of problem analysis on their own, including determining how to simulate selection and iteration statements, how to handle and report syntax errors, etc.

SOLUTION DESIGN:

Acceptable methods for implementing a small interpreter are discussed in class, and the specification for the syntax of the language is provided, but the students design their own methods for satisfying most of the specification requirements, and typically there is much individual variation in the students' responses to those requirements.