

# Syllabus – Spring 2008

## CS 350 Programming Language Design

**CRN 22106:** TR 3:00PM - 4:15, KT 250

---

### **Instructor**

Mark Temte, Ph.D.

Office: ET-125F

Office hours: MW 4:30 PM – 5:45, TR 1:30 PM – 2:45

Phone: 260-481-6181

Email: [temte@ipfw.edu](mailto:temte@ipfw.edu)

Course web site: [users.ipfw.edu/temte/](http://users.ipfw.edu/temte/)

### **Course Description**

**P:** 260 and 271. A survey of language design issues and their implications for translation and run-time support. Examination of modern programming languages and features: abstract data and control structures, procedures, parameter passing mechanisms, block structuring and scope rules, input/output, and storage management. Models of run time behavior. Comparison of imperative and declarative programming languages.

### **Course Learning Outcomes**

(These support the ABET Program Outcomes found at the end of the syllabus, as indicated by the letters in parentheses.)

Upon successful completion of the course requirements, a student should be able to:

1. Show knowledge of the evolution of modern programming languages (a).
2. Demonstrate facility with BNF for specifying programming language syntax (a, i).
3. Specify various control structures using operational semantics (a, i).
4. Compute weakest preconditions using the principles of axiomatic semantics (a, i).
5. Show understanding of issues involving variables: data types, binding, strong typing, and scope (a, b, i, j).
6. Show understanding of various subprogram parameter passing methods (a, b, i, j).
7. Show understanding of the issues involved with reference types: handling pointers, lost heap-dynamic variables, and garbage collection (a, b, i, j).
8. Evaluate design trade-offs associated with various language features (a, c).
9. Design and implement an application in Smalltalk using a number of cooperating classes (b, c, i, k).
10. Design and implement a lexical analyzer and recursive-descent parser for a simple language (b, c, i, k).
11. Implement an application using the functional language Scheme (b, c, i, k).

12. Implement an application using predicates in the logic-programming language Prolog (b, c, i, k).

**Required textbook**

Robert W. Sebesta, *Concepts of Programming Languages (8<sup>th</sup> edition)*, Addison Wesley, 2006. ISBN 978-00321-49362-0.

**Exams**

There will be two 75-minute exams and a final exam (see tentative schedule below). No make-ups will be given unless approved in advance or in case of a valid emergency. In the latter case, contact the instructor immediately.

**Projects**

Approximately four projects will be assigned using various languages. Each project is due at the beginning of class on the due date. A 10% penalty will be imposed the first day a project is late and an additional 5% penalty for each day thereafter (Monday through Friday only). All implementations must adhere to the established standards and must be your own work and not the result of plagiarism. In particular, group work on projects is not permitted. Violation of this policy will result (at minimum) in a project grade of zero.

**Attendance**

Attendance is a University requirement, and you are expected to attend every class. Your grade may be adversely affected by any absences. Notify the instructor by telephone or email if you cannot attend.

**Grading policy**

Projects 40%, 75-minute exams 20% each, final exam 20%. The score used in determining the final grade will be no higher than one letter grade above that derived from tests alone. (Unless curved, A: 90-100, B: 80-89, C: 70-79, D: 60-69, F: below 60.)

**Special needs**

If you have a disability or acquire one, contact the Director of Services for Students with Disabilities (Walb 113, telephone number 481-6658) for services and accommodations available at IPFW. For more information, visit the web site for SSD at <http://www.ipfw.edu/ssd/> .

## Tentative schedule

Week of:	Topics	Chapter	Comments
Jan 14	Introduction	1	
1 2	Evolution of programming languages	2	No classes Monday
8 2	Smalltalk, ADTs, and OOP	Tutorial 11,12	
Feb 4	Syntax and semantics	3	
1 1	Syntax and semantics	3	
8 1	Lexical and syntax analysis	4	Test 1: Thursday, Feb 21
5 2	Lexical and syntax analysis	4	
Mar 3	Names, bindings, type checking, and scopes	5	
0 1			Spring break week
7 1	Functional programming languages: Lisp, Scheme, and Haskell	15	Last day to withdraw is Friday
4 2	Data types	6	
1 3	Expressions and assignment statements	7	Test 2: Thursday, April 3
April 7	Logic programming and Prolog	16	
4 1	Statement-level control structures	8	
1 2	Subprograms and subprogram implementation	9,10	
8 2	Exception and event handling	14	
May 5	Final Exam: Thursday, May 8		

	10:30 AM - 12:30, KT 250		
--	--------------------------	--	--

## **ABET Program Outcomes**

(These are among the criteria ABET, our accrediting agency, uses in accrediting degree programs in computing.)

- a. An ability to apply knowledge of computing and mathematics appropriate to the discipline.
- b. An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution.
- c. An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs.
- d. An ability to function effectively on teams to accomplish a common goal.
- e. An understanding of professional, ethical, legal, security and social issues and responsibilities.
- f. An ability to communicate effectively with a range of audiences.
- g. An ability to analyze the local and global impact of computing on individuals, organizations, and society.
- h. Recognition of the need for and an ability to engage in continuing professional development.
- i. An ability to use current techniques, skills, and tools necessary for computing practice.
- j. An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.
- k. An ability to apply design and development principles in the construction of software systems of varying complexity.