CSCI 365 SYLLABUS

| | |
|---|---|
| **COURSE NAME, NUMBER** | Programming Language Concepts, CS 365 |
| **SEMESTER, YEAR** | Spring 2010 |
| **INSTRUCTOR** | T. VanDrunen |
| **OFFICE / TELEPHONE / EMAIL** | Armerding 112 752-5692 Thomas.VanDrunen@wheaton.edu |
| **OFFICE HOURS** | MWF 3:15–4:45 pm; Th 9:00-11:30 am |
| **COURSE WEBSITE** | `http://csnew.wheaton.edu/~tvandrun/cs365` |

**RESOURCES** Tucker and Noonan, *Programming Languages: Principles and Paradigms,* (Second Edition). McGraw Hill, 2007.

**COURSE DESCRIPTION**

Formal definition of programming languages, including syntax and semantics; recursive descent parsing, data structures, control constructs, recursion, binding times, expression evaluation, compiler implementation; symbol tables, stacks, dynamic allocation, compiler compilers.

**GOALS AND OBJECTIVES**

1. Students will use formal models to specify, describe, and reason about programming languages and their features.
2. Students will implement programming languages by writing parts of analyzers, interpreters, and compilers.

**ASSESSMENT PROCEDURES**

1. Problem sets will exercise students' abilities to use language models and write proofs of language attributes.
2. Projects will teach students to apply their understanding of programming languages in building programming language systems.
3. The midterm and final exam will evaluate students' mastery of the comprehensive material.

**Grading:**

| | *weight* |
|---|---|
| Assignments | 10 |
| Projects | 50 |
| Midterm exam | 20 |
| Final exam | 20 |

**General explanation and goals.** This course is an even-handed mix of theory and practice. On the theory side we talk about formal ways to talk about languages: Grammars help us to specify what programs are permissible in the language, and semantic models help us to specify what a program means. These allow us to focus on the language itself as a computational system apart from any particular machine or implementation. Moreover, in some cases this allows us to prove things about a language and programs in the language—for example, such and such a language has a type system strong enough that we can check a program for type errors at compile time and guarantee that certain errors cannot happen at runtime.

On the practical side, you'll be doing a lot of programming. The projects will involve writing small language systems—analyzers, interpreters, and compilers. In most cases, these will allow you to execute programs in the languages we'll look at in class. It's a whole lot of fun.

In a typical couple of class periods, you will (1) see a new language feature or idea introduced with a new small language that supports it (in lecture), (2) see a formal description of the feature using semantic models (in lecture), (3) extend or refine the semantic rules (in a short assignment), (4) see an implementation of the feature/language in Java or ML (in lecture), and (5) write your own implementation in Java or ML, whichever we didn't using in class (in a project).

**What this course is not.** This is not a "comparative programming languages" course. We won't be dabbling in a long list of classic old languages or fancy new ones. We don't have a course like that because (1) that sort of thing is built into other places of the curriculum, (2) in ten years, there will be all new languages being used, (3) if you ever need to learn a new language for a job or research project, you should be able to learn it on your own fairly well, and (4) I don't think it's as interesting.

## SPECIAL EXPECTATIONS

### Academic Integrity

A course of this class size and difficulty may present situations where the line between beneficial collaboration and cheating is blurry. To clarify, we make a distinction between *daily work* and *projects*.

At the end of most class periods I will assign a few small problems for you to work on for the next class period. The purpose is to reinforce the material, to prepare you for the next period's discussion, and to show you what to expect on the exams. Daily work has a minimal affect on your grade for the course. Thus collaborating on daily work usually will be acceptable—judge for yourself whether you are benefiting from it.

Most of your work in this class will be in the programming projects, and they contribute to 50% of your course grade. These are to be your own work. You may discuss the problem with your classmates and you are encouraged to share testcases. Your code, however, must be solely your own work.

### Late projects

You are allowed a total of three days during the course of the semester, which may be divided up (in whole-day units) among the projects in any way. Other late projects and assignments will not be accepted. *Please notify me if you are turning an assignment in late; this helps me plan grading.*

### Attendance

While I was an undergraduate, I missed a grand total of two classes, and one of them was to take the GRE. I expect the same from my students. Since being a student is your current vocation, since your learning now will affect your ability to support a family and church later in life, and since you, your family, and/or a scholarship fund are paying a large sum of money to educate you, being negligent in your schoolwork is a sin. I do not take attendance, but I do notice. When missing a class is unavoidable, it is courtesy to inform the instructor, ahead of time if possible.

### Special needs

Whenever possible, classroom activities and testing procedures will be adjusted to respond to requests for accommodation by students with disabilities who have documented their situation with the registrar and who have arranged to have the documentation forwarded to the course instructor. Computer Science students who need special adjustments made to computer hardware or software in order to facilitate their participation must also document their needs with the registrar in advance before any accommodation will be attempted.

### Office hours

In the past I have tried to keep an "open-door" policy, that is, I don't mind if you stop by even when it is not my office hours. However, to help manage my load this semester, I've decided to reserve Tuesdays for uninterrupted work. Accordingly, *please to do not come to my office on Tuesdays without an appointment.* If you need help on a Tuesday, please make an "appointment" by sending me an email asking something like, "May I stop by and ask a question right now?" I will likely say yes. (Also, any time my door is closed, it means I'm doing something uninterruptable, such as making an important phone call. Rather than knocking, please come back in 5 minutes or send me an email.)

**Examinations.** There will be a midterm (currently scheduled for Wednesday, Mar 3). The final exam is Tuesday, May 4, at 1:30 PM. I do not allow students to take finals early (which is also the college's policy), so make travel appropriate travel arrangements. The final will not be "explicitly" cumulative, which means there will not be any questions designed solely to test A-quad material. However, B-quad material builds heavily on A-quad material.

**Projects.** Most of the work outside of class will be on programming projects, of various lengths. All of these will involve writing part of a programming system (that is, an interpreter, analyzer, or compiler) for a language based on the textbook's example language, Jay, or a similar language. As mentioned above, these should be worked on independently.

**Outline.**

I. Prolegomena
- A. History of programming languages
- B. Basic terms and concepts

II. Imperative programming languages
- A. Lexical and syntactic strcuture
- B. Parsing; concrete and abstract syntax
- C. Semantics
  1. Types and type-checking
  2. Formal semantics (denotational and operational)
- D. Implementation
  1. Interpretation
  2. Compilation (to be revisited at the end of the course)
  3. Source-to-source translation
- E. Features
  1. Floating point and pointer types
  2. Procedures
  3. Arrays
  4. Records
- F. Alternative features

III. Object-oriented programming languages
- A. Objects and classes
- B. Types and polymorphism
- C. Inheritance
- D. Exceptions
- E. Alternative formulations

IV. Functional programming languages
- A. The lambda calculus
- B. Efficiency
  1. Tail form
  2. Continuation-passing style
- C. Types
  1. Type-checking
  2. Proving type-soundness

V. Compilation

VI. Alternative programming paradigms
- A. Data flow programming languages
- B. Declarative programming languages
- C. Concurrent programming languages