CSCI 365

Programming Language Concepts

Spring 2012 MFW 9:15–10:20 am SCI 131

http://cs.wheaton.edu/~tvandrun/cs365

Thomas VanDrunen

☎630-752-5692 630-639-2255 ✉Thomas.VanDrunen@wheaton.edu

Office: SCI 163 Office hours: MWF 2:00–4:00 pm; Th 9:30–10:45 am.

Contents

CATALOG 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.

EXPLANATION. The catalog description is pretty much a vague list of terms that could be relevant in a programming languages course. It doesn't define the course very well, and I don't follow it much.

There are three kinds of "programming languages" courses: Courses on *comparative programming languages* where students learn a smidgen of many languages, comparing and contrasting how various features are realized and the style of programming that each encourages; courses on *compiler construction* where students learn the pragmatics of how to implement a programming language; and courses on *programming language design* where students study the theory of advanced features and their implications for implementation.

This course 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.

> *I often feel that the American programmer would profit more from learning, say, Latin than from learning yet another programming language.* —Edsger W Dijkstra, EWD611

Instead, this course steers a middle course between the other two models, giving an even-handed mix of theory and practice. We will study formal models for specifying language features and reasoning about them. We will write many small analyzers, interpreters, and compilers to consider the implementation of language features close-up.

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

OBJECTIVES. The chief goal of this course is to understand how the features of a programming language are specified and implemented and how constraints on the models and systems effect programming language design decisions. Subordinate goals include

- using formal models to specify, describe, and reason about programming languages and their features.
- implementing programming languages by writing parts of analyzers, interpreters, and compilers.

THEMES. A variety of themes are interwoven among the topics of this course and drive the decisions about what to cover.

Programming paradigms. The old way of categorizing programming languages into paradigms (like imperative, object-oriented, and functional) is showing its age. Newer languages tend to encorporate features from each. Even though it is not as useful anymore for categorizing *languages*, it still is handy for categorizing *features*, as well as programming *styles/mindsets* (the program is a list of instructions / set of interacting objects / composition of functions).

We will use subsets of Java to study imperative and object-oriented features and subsets of ML to study functional features.

Formalization. We will consider various models for defining the semantics of a langauge, at different levels of abstraction.

Implementation. We will consider various ways to implement programming languages, generally: interpretation (write a program that executes programs in a language, modelling a machine at various levels of abstraction or following a formal model at various levels of consistency); source-to-source compilation (write a program that translates programs from one language to another language); and compilation (write a program that translates programs from one language to the machine language for a real or virtual machine).

Static analysis. We will consider what analyses can be performed on a program in a language without running the program to screen for certain errors or make optimization decisions. (Type-checking is the most important example of static analysis.)

OUTLINE. See the course website for schedule of these topics.

I. Prolegomena
   A. History of programming languages
   B. Philosophy of programming languages
   C. Defining programming languages

II. Imperative programming
   A. Lexical and syntactic structure
   B. Parsing algorithms
   C. Concrete and abstract syntax
   D. Types and type-checking
   E. Formal semantic models
   F. Implementation of features
      1. Loops, blocks, and scope
      2. Switch statements
      3. Casting
      4. Floating point
      5. Procedures
         a. Parameter passing modes
         b. Nested procedures
      6. Arrays and dynamic memory
      7. Records (structs)
      8. Exceptions

III. Object-oriented programming
   A. Basic model
   B. Types, subtypes, and polymorphism
   C. Inheritance
   D. Alternate formulations

IV. Functional programming
   A. Foundations; the lambda calculus
   B. Building a functional language
   C. Tail form and continuation-passing style
   D. Types and type soundness proofs

V. Compilation

VI. Alternatives
   A. Data flow languages
   B. Declarative languages
   C. Concurrent languages

Course procedures

HOW WE DO THIS COURSE. The typical rhythm for this course is (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).

Readings. The textbook is mainly for you to have a written resource to which to compare class discussion. It is not a perfect fit for our course, but no textbook that I know of is. Appropriate sections for each class period can be foud on the course website. Occasionally you will be asked to read a specific section to prepare for class or in conjunction with a short assignment.

Assignments. Assignments range in size and difficulty. Most, especially the small ones, will be graded for completeness and are intended to prepare you for the next class. Some, however, will be graded on correctness.

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 subset of Java or ML; the programming system itself will be written in Java or ML, depending on the project. These should be worked on independently. See the course website for approximate assignment and due dates. Note that some projects will have overlapping timeframes.

Examinations. There will be a midterm examination and a final exam. The final exam will be semi-cumulative: it will be designed to test material in the second half of the course, but necessarily will depend on material from earlier. Both will include both a written portion and a programming portion. The midterm exam will be a take-home exam to be taken within a proscribed length of time (probably two hours) at some point during the week of Feb 27–Mar 2. Note that projects 5 and 6 will also be due Mar 2. Class will be cancelled on Feb 29 to provide more time for finishing the projects and midterm.

Grading. Note that the final exam is Tuesday, May 1, at 1:30 PM.

instrument	*weight*
Assignments	10
Projects	50
Midterm exam	20
Final exam	20

Policies etc

Academic integrity. Projects must be done independently (as, for example, in CSCI 235 or CSCI 245). Students may collaborate on assignments unless instructed otherwise. Failure to comply may result in point deduction or rejection of the project or assignment altogether.

Late assignments. 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. (Note this is one more late day than I allow in CSCI 235 or CSCI 245.) 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. Students are expected to attend all class periods *on time*. It is courtesy to inform the instructor when a class must be missed.

Examinations The midterm is scheduled for the week of Feb 27–Mar2. The final exam is Tuesday, May 1, 1:30 PM. I do not allow students to take finals early (which is also the college's policy), so make appropriate travel arrangements.

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. I try to keep a balance: Stop by anytime, but prefer my scheduled office hours. Monday, Wednesday, and Friday my office hours are 2:00–4:00 pm, except that *on Wednesday they end 15 minutes early, at 3:45 pm*; on Thursday my office horus are 9:30-10:45 am. I do not have office hours on Tuesday.

If these times do not work for you, you can schedule an appointment (your best luck would be some other time Thursday morning or after lab). 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 a few minutes or send me an email.

Dress and deportment. Please dress in a way that shows you take class seriously—more like a job than a slumber party. (If you need to wear athletic clothes because of activities before or after class, that's ok, but try to make yourself as professional-looking as possible.) If you must eat during class (for schedule or health reasons), please let the instructor know ahead of time; we will talk about how to minimize the distraction.

Electronic devices. Please talk to me before using a laptop or other electronic device for note-taking. I will discourage you from doing so; if you can convince me that it truly aides your comprehension, then I will give you a stern warning against doing anything else besides note-taking. Trying out programming concepts on your own during class time is not productive because it takes you away from class discussion; that is what lab time is for. You cannot multi-task as well as you think you can. Moreover, please make sure other electronic devices are silenced and put away. ***Text in class and DIE.***